SCHOLAR Study Guide

SQA Higher Computing
Unit 2: Software Development

David Bethune
Andy Cochrane
Tom Kelly
Ian King
Richard Scott

Heriot-Watt University

Edinburgh EH14 4AS, United Kingdom.

First published 2004 by Heriot-Watt University.

This edition published in 2011 by Heriot-Watt University SCHOLAR.

Distributed by Heriot-Watt University.

SCHOLAR Study Guide Unit 2: Higher Computing

1. Higher Computing

ISBN 978-1-906686-60-4

Printed and bound in Great Britain by Graphic and Printing Services, Heriot-Watt University, Edinburgh.

Acknowledgements

Thanks are due to the members of Heriot-Watt University's SCHOLAR team who planned and created these materials, and to the many colleagues who reviewed the content.

We would like to acknowledge the assistance of the education authorities, colleges, teachers and students who contributed to the SCHOLAR programme and who evaluated these materials.

Grateful acknowledgement is made for permission to use the following material in the SCHOLAR programme:

The Scottish Qualifications Authority for permission to use Past Papers assessments.

The Scottish Government for financial support.

Contents

Topic 1

The Need for the Software Development Process

Contents

Prerequisite knowledge

There are no prerequisites for this introductory topic.

Learning Objectives

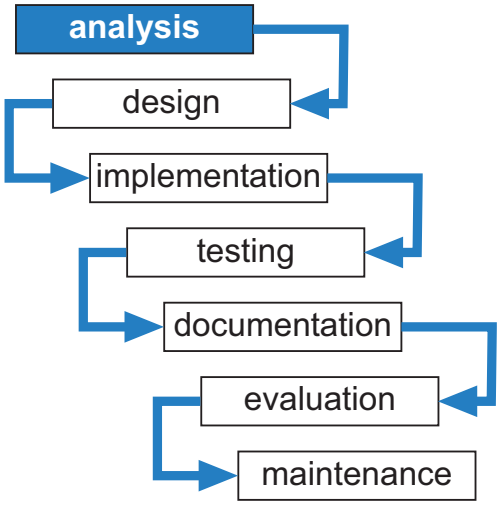
- *to understand what is meant by 'software development'*
- *to recognise the professional issues that are involved in the development of a computer system*
- *to understand the need for software development*
- *to understand what is meant by well planned programs.*

1.1 Introduction

Computer systems are now an important aspect of everyday life. It would be difficult to visualise a world without the day to day impact that computers have on our lives: cash machines, supermarket tills, petrol pumps, travel tickets, payslips and bills, the Internet and e-commerce and so on.

The first few topics of this unit describe the development of software. The development of software follows a definite process, known as the **software development process** (SDP).

As you should already know, this process can be split into 7 stages:



By following these 7 stages, it should be possible to design and build software systems that are:

- of high quality
- cost effective
- produced to specification
- delivered on time

The idea is to bring together a variety of tools, techniques and methods which will help create **reliable** software. Programs can be designed to be **maintainable** - that is they are presented and documented so clearly that they can be updated by another programmer at some future time.

There are many reasons why the software development process is so important. These include:

1. increased complexity and sophistication of computing systems
2. escalating costs of software systems
3. unreliable software systems
4. the cost of maintaining software

Following the software development process is especially important in the development of "real life' programs. If you understand the problems of large-scale programming, the need for the software development process will be obvious. Without this understanding, it is all too easy to dismiss the software development process as a not-very-useful exercise, but one which will give you a Higher credit.

1.2 Real-life programs and classroom programming

What are the differences between the two? Table 1.1 should give you an indication.

Table 1.1: Real programs vs classroom programs

	Real-life Programs	**Classroom programs**
Scope	. . . are large and complex. It is difficult to hold their details in your mind. Because of this you have to specify them formally.	. . . are small and simple. They can be described in a few sentences and probably understood in minutes.
Difficulties	The main difficulty is with (a) understanding the problem; (b) ensuring that your understanding actually matches the users' requirements; (c) designing a solution.	The main difficulty you have is with writing the solution in a programming language which you are just learning, and getting your program to work correctly.
Development time	Development time is long. Maybe as long as 5 years. It is impossible to remember all aspects of a program for this length of time, so you have to document the design decisions and all your changes.	Development time is short. You should be able to get the core of the solution working in a few hours. Consequently you can hold details of all aspects of the program in your head. Documentation is skimpy or non-existent.
Testing	Testing is going to be long, intensive and exhaustive. It will be done by a separate team whose function is solely to test programs and find faults.	Your testing is likely to be fairly basic.
Lifespan	A program's lifetime may be decades, and it may undergo changes (called 'maintenance'). The maintenance is probably going to be done by someone who has not written the original program and knows nothing about the original design decisions unless they have been documented.	The life of the program is short - only long enough for you to pass! Similarly, the documentation only has to stand up to the examiner's scrutiny.
Uses	Programs are written for use by other people who might have little understanding of computers.	The programs are not intended to be used.

Real-life programs are written for use by **clients** who require the software systems to help them do their jobs. They probably expect them to work like any complicated electrical appliance such as a TV. You switch it on and it works. Change channels and the channels change. The brightness and other controls do what they are supposed to do, without stopping the TV from working or having some unexpected side-effects.

The programmers' goal is to produce software as **robust** and easy to use as a TV. In reality this is going to be impractical, but when you design programs this should be something you need to bear in mind.

For example, keyboard input should be checked and incorrect input rejected. But this is not enough; you need to give a message to the user to help them type the correct information next time. If you do not do this, you leave them floundering; they know what they typed is wrong, but have no idea why or what the correct input should be.

1.3 Computing disasters

It has been said *"Computers make very fast, very accurate mistakes"*.

The software development process is important because without it you are bound to have a disaster in a project of any great size. Large projects demand a large investment of money, time and resources before any returns are possible.

Consider the following:

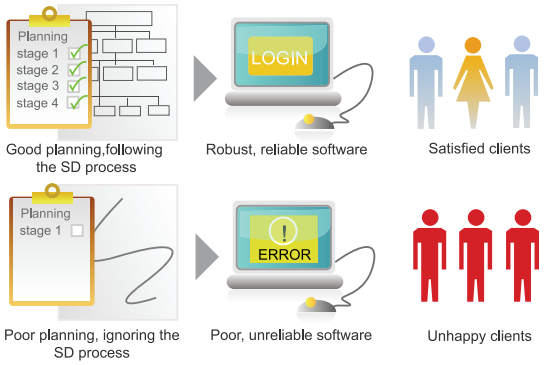
- One in five software systems fail to deliver to the agreed specification
- Software maintenance is the single highest computer-related cost for many companies
- Some systems can take many years to develop running the risk of obsolescence before they are even commissioned.

1.3.1 Examples of Computing Disasters

The list below contains a few examples of computing disasters that could be attributed to complexity and the need for better planning:

1. **Ariane 501** - on the 4th June 1996 Ariane 501 exploded 40 seconds after takeoff due to a software bug. This was an incidence of data conversion of a too large number!
2. **Y2K Problem - The Millennium Bug** - problems with the way the date was stored on computers
3. **NASA Mars Lander** - problems in software occurred when different measurement units were used - there was confusion between pounds and kilograms
4. **Home Office Immigration System** - Ordered in 1996 and supplied eighteen months late by Siemens in 1999 at a cost of £80 million. It was supposed to speed the processing of asylum claims up but could not cope with the backlog and did the very opposite. It was finally scrapped in 2001. Government ministers were blamed for ordering the "over-complex" system in the first place
5. **London Ambulance Service** - in 1992 they took on a despatch system that failed calamitously. Ambulances were sent to the wrong place, did not arrive when expected and the system generally caused major disruption to patient care and services
6. **Passport Agency** - in 1999 a new system (Siemens again) that worked much less efficiently than the system it replaced. Delays went up from two weeks to seven, with a backlog of more than half a million passports. This cost the taxpayers £12 million and forced many people to cancel holidays. The Government increased passport charges to recoup wasted money from this system
7. **Post Office swipecard system** - a one billion pound project that ICL were to install in throughout Britain. It began in 1996 and was stopped in 1999, having been, as the official report said, "blighted from the outset"
8. **French national library** - received a new system (two years late and 40% over budget) which worked so badly that librarians walked out
9. **ATMs in Germany** - during the changeover from Marks to Euros in 2002 a programming error in the banking system allowed people to withdraw any amount cash by typing in an arbitrary PIN code.
10. **London Millennium Bridge** - in 2000 the newly built Millennium Bridge wobbled when pedestrians attempted to walk over it. In the computer simulation the programmers had used the wrong estimates for pedestrian forces.

Of course, all these systems were planned, but with better planning, these mishaps might have been prevented. Without planning, probably all large projects would go wrong.



The most basic notion of well planned programs is producing software which does what the user wants. This is easy to say, but more than a little difficult to do. If you are taking this course at the same time as learning a programming language, then we are sure you know this is so from your own experience.

1.3.2 Information sources on computing disasters

The World Wide Web and Internet links go out of date rather quickly. You might want to try accessing the following sites which are current at the beginning of 2004.

1. **Ariane 501**
   - A Bug and a Crash by James Gleick, http://www.around.com/ariane.html
   - Ariane-5: Learning from Flight 501 and Preparing for 502 - European Space Agency (ESA), http://esapub.esrin.esa.it/bulletin/bullet89/dalma89.htm
2. **Y2K Problem - The Millennium Bug**
   - What is the millennium bug? - http://www.wisegeek.com/what-is-the-millennium-bug.htm
3. **NASA Mars Lander**
   - Metric mishap caused loss of NASA orbiter - CNN, http://www.cnn.com/TECH/space/9909/30/mars.metric.02/
4. **Passport fiasco**
   - http://news.bbc.co.uk/1/hi/uk_politics/486821.stm
5. **Asylum issue**
   - http://news.bbc.co.uk/1/hi/uk_politics/1171147.stm
6. **Millennium bridge**
   - http://www.arup.com/MillenniumBridge/

Investigating computing disasters

Several WWW sources of information about computing disasters have been given above. Try to find out more about any two (or more) that might interest you. Write a brief summary of two disasters that you have found.

1.4 Well planned programs

The most basic notion of well planned programs is to produce software that does what the user wants.

Other characteristics of well planned programs are:

1. the software should be **maintainable**: software with a long lifetime is likely to need changing. This means that software must be written and documented in a way which makes change simple and straightforward
2. the software should be **reliable** and robust
3. the software should be **efficient**. It should not make unreasonable demands on the hardware on which it will run. But higher efficiency can lead to less maintainable software as programmers use shortcuts which are effective, but difficult to understand
4. the software should have an appropriate user interface. 'Appropriate' means that you need to consider the background and capabilities of intended system users.

There is often a trade-off between these factors and the cost of producing the software. This is illustrated in the diagrams in Figure 1.1.

As you can see, costs increase dramatically as developers attempt to provide software that is 100 percent reliable. Making software more efficient also increases the amount of money that needs to be invested.

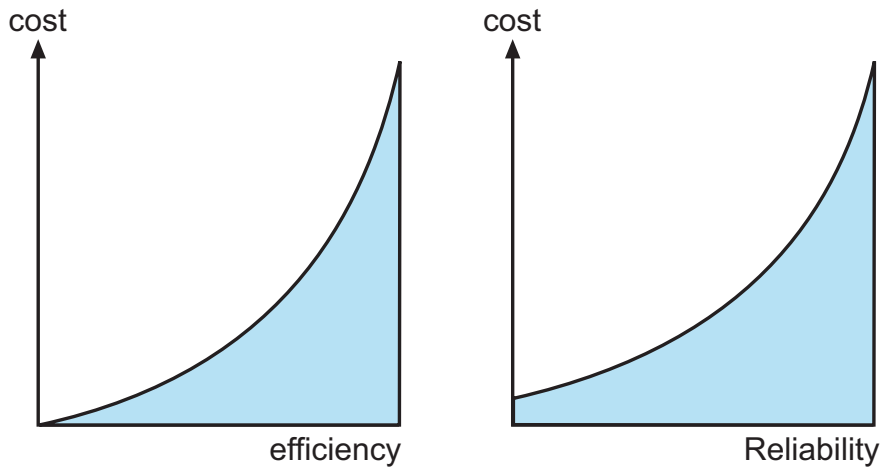


Figure 1.1: Cost of Software Systems

1.5 Summary

The following summary points are related to the learning objectives in the topic introduction:

- the software development is a complex process involving many facets of professional expertise;
- no matter how simple or complex a program is, the elements of software development are almost mandatory to ensure well-planned programs;
- no matter how rigorously the software development process is followed, errors can still occur in large, commercial software systems.

1.6 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 2

Features of Software Development Process

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe the following stages of the software development process, analysis, design, implementation, testing, documentation, evaluation, maintenance;*
- *describe and be able to use test data (normal, extreme and exceptional);*
- *describe the features of a user guide and a technical guide;*
- *evaluate software in terms of fitness for purpose, user interface and readability.*

Learning Objectives

After studying this topic, you should be able to:

- *understand the iterative nature of the software development process*
- *describe each stage of the software development process*
- *explain the purpose of the software specification and its status as a legal contract*
- *understand and be able to describe corrective, adaptive and perfective maintenance*
- *explain the need for documentation at each stage of the software development process*

Revision

Q1: The software development process consists of seven stages. Three of the stages are analysis, testing and maintenance. Which one of the following statements correctly identifies the missing stages, in order?

a) Design, documentation, evaluation, implementation
b) Design, evaluation, documentation, implementation
c) Design, implementation, documentation, evaluation
d) Design, documentation, implementation, evaluation

Q2: Which one of the following is an essential item of documentation that should be produced during the software development process?:

a) Design notation
b) Test data
c) User interface
d) User and Technical Guides

Q3: Which one of the following is NOT an aspect of the user interface of a program?:

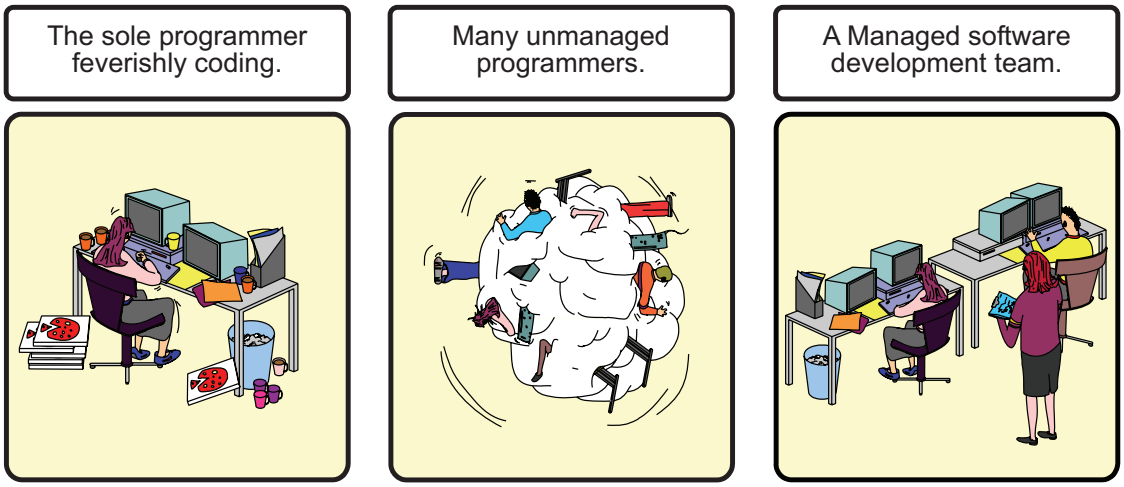
a) Program listing
b) Help screens
c) Instruction screens
d) Screen layout

2.1 Introduction

This topic considers the software development process. From the initial client specification to the production of a working program can take considerable time and effort by the development team. The process involves constant revision and evaluation at every phase which makes it an iterative process. This ensures quality and efficiency in the final product. Various models are introduced that aid the software development process but you will find that the perfect solution does not exist.

An individual may write a program for personal use. If it does not work then it can be changed. If more features or facilities are required then the individual can make amendments to their program.

This *ad hoc* method is not satisfactory in commercial environments where the goal is the creation of large scale software. A more structured approach is necessary.



Organisations creating software usually do so for profit. Money, time, and people are involved. The people involved have different points of view:

- some are clients, wanting to buy software;
- some are developers, concerned in creating software;
- managers are concerned with efficiency and profit within their organisation.

In the development of software, the three aspects which the developer must consider are:

- data
- processes
- human-computer interface

In traditional structured design, the primary tasks are to focus on the processes. A **process** is the work that a program carries out on data or in response to certain inputs.

The software development process does not always start in the same place. Sometimes there will only be an outline of the problem. At other times, a **specification** will be available.

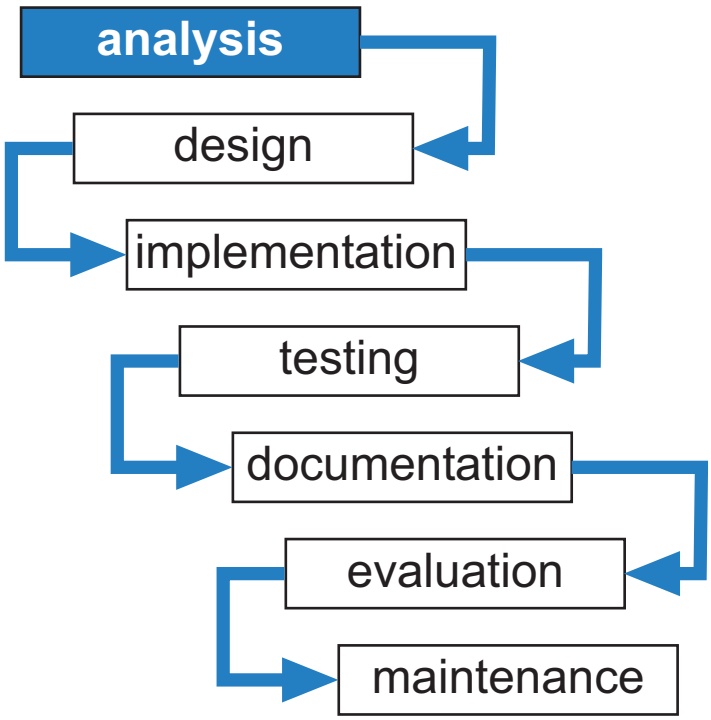
The specification must be agreed with the clients. Work on the problem and the solution is often carried out by a group of people, called the **development team**.

The aim of the team is to produce a new software system that will solve the problem.

2.2 The need for iteration

The traditional software development process contains a sequence of stages. The precise names of the stages, and even the number of stages, are not universally agreed. They differ from book to book and from developer to developer. Much depends on the aspects of software development that a book wishes to emphasise or that a particular developer prefers.

In this course we will consider the following stages:



The idea here is that the development of the system flows down or cascades through the stages like water flowing down a fall. As each stage is completed responsibility and control is passed down until the final section is completed.

The process, as it stands, represents an ideal world rather than reality. Developers do not know everything the client will need at the start of a project; they make wrong decisions, possibly based on incomplete information.

The model can be improved significantly if it is made to be **iterative**.

Ideally, you would start a process with the analysis and work through the stages in turn, doing everything only once. In practice, this happens rarely. People make mistakes, faults become apparent that can only be corrected by going back to an earlier stage of the process.

An iterative development model

There is an on-line illustration of the need to use an iterative development model. You should view this animation now.

2.2.1 Review questions

Q4: Which one of the following statements regarding the order of the stages in the software development process is correct?

a) Design, implementation, testing, documentation
b) Analysis, testing, implementation, design
c) Design, testing, evaluation, implementation
d) Analysis, design, testing, evaluation

Q5: The software development process may involve iteration. Which one of the following statements is true if iteration is used?

a) The team may go back to an earlier stage to deal with a problem.
b) The software is guaranteed to be error free.
c) Each stage is carried out twice.
d) Once all the stages have been completed, the team go back and start again.

Q6: Software developers cannot get the software correct at the first attempt. This is because:

a) Software systems can be very complex
b) Unforeseen errors always creep in that take time to solve
c) There could be problems with the client changing his/her mind
d) All of the above

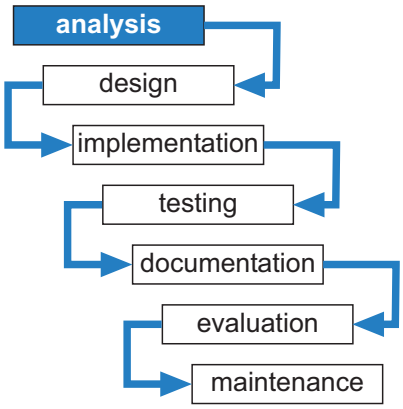
Q7: Regular feedback of information to members of the development team is important. This is done in order to:

a) Speed up the development process
b) Enable personnel to discuss progress
c) Keep the team happy
d) Lower the costs on a regular basis

Sentence completion - software development process

On the Web is a sentence completion task on the software development process. You should now complete this task.

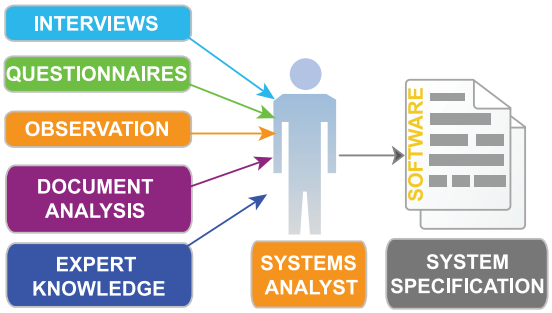
2.3 The Analysis Stage



This stage is crucial to the entire cycle of events. Any problems occurring at this stage will be propagated through the system and will become increasingly costly to rectify when discovered.

Analysis is an attempt to understand a given problem, clearly and exactly, in order to generate a solution. The outcome will be a specification that is used as the basis for all subsequent work.

Analysis, sometimes called systems analysis, is the job of a specialist person - the systems analyst.

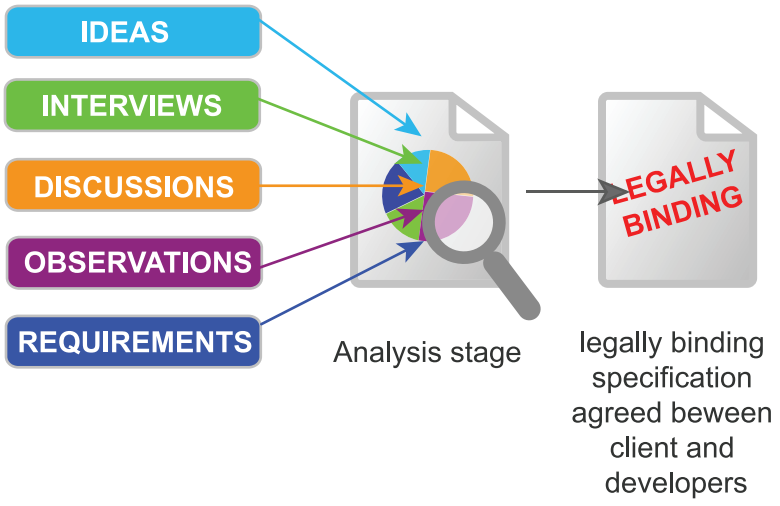


Full systems analysis has three phases:

1. collection of information
2. analysis of information collected
3. production of a problem specification or user requirements specification.

Sometimes, this stage begins with a vague idea or rough outline of the problem. On other, rather more formal, occasions, it will start with a full **requirements specification**. This will include:

- hardware and software specifications,
- notes on project issues such as,
  - objectives,
  - constraints,
  - costs
  - and schedule.
- It may also include a full **functional specification**, which will describe exactly how the system is meant to behave. The functional specification is what the development team will follow in creating the software system.

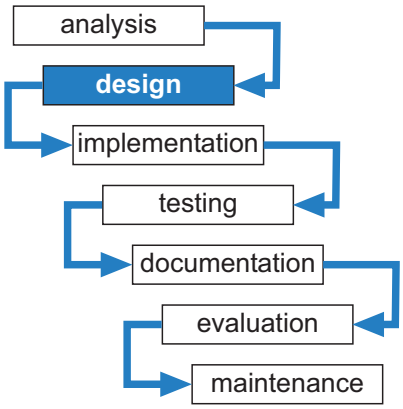


Questions to be asked at this stage would include:

- What are the new system requirements?
- What are the costs involved?
- How long will it take to implement?

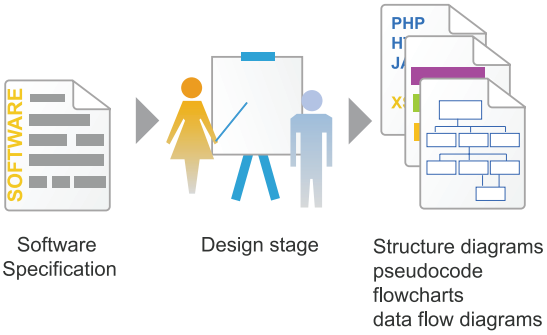
Details are gathered by a variety of methods such as interviews and questionnaires.

2.4 The Design Stage



The product of the analysis stage is the completed specification. During the design stage, the specification of the problem is used as the basis for a planned solution.

The work on the design is carried out by a designer: this might be same person as did the analysis or it may be a member of the programming team. Most analysts can program and many programmers can carry out analysis. People who can do both are described as analyst/programmers.



The design process is methodical, using techniques such as structure charts and pseudo-code. The problem is approached by breaking it down into a collection of relatively small and simple tasks. This approach is known as top-down design with stepwise refinement.

The development team will attempt to make the design of the program both robust and reliable.

- A reliable program is one that does not stop because of faults in its design
- A robust program is one that can cope with errors when it is running

To put it another way, an unreliable program is one that hangs or crashes for no apparent reason whereas a non-robust program is one that cannot cope with events that the world

throws at it.

Identifying the characteristics of good software design

On the Web is an interactivity. You should now complete this task.

2.4.1 Review questions

Q8: Which one of the following processes describes breaking a complex system down into more manageable components?

a) top-down design
b) using pseudocode
c) refined design
d) prototyping

Q9: In the software development process which of the following is a legal contract?

a) functional specification
b) problem specification
c) requirements specification
d) software specification

Q10: Which of the following are identified during the analysis stage of software development?

a) the main costs of the project
b) time taken to complete the project
c) hardware required to run the system
d) All of the above

Q11: The completed software system should be able to cope with many errors while running. This means that the software is (choose one):

a) Portable
b) Robust
c) Reliable
d) Stable

Q12: Which of the following is included in the **functional specification**?

a) a description of what the software must do
b) the hardware used to run the software
c) the nature of the problem to be solved
d) an outline of the problem solution

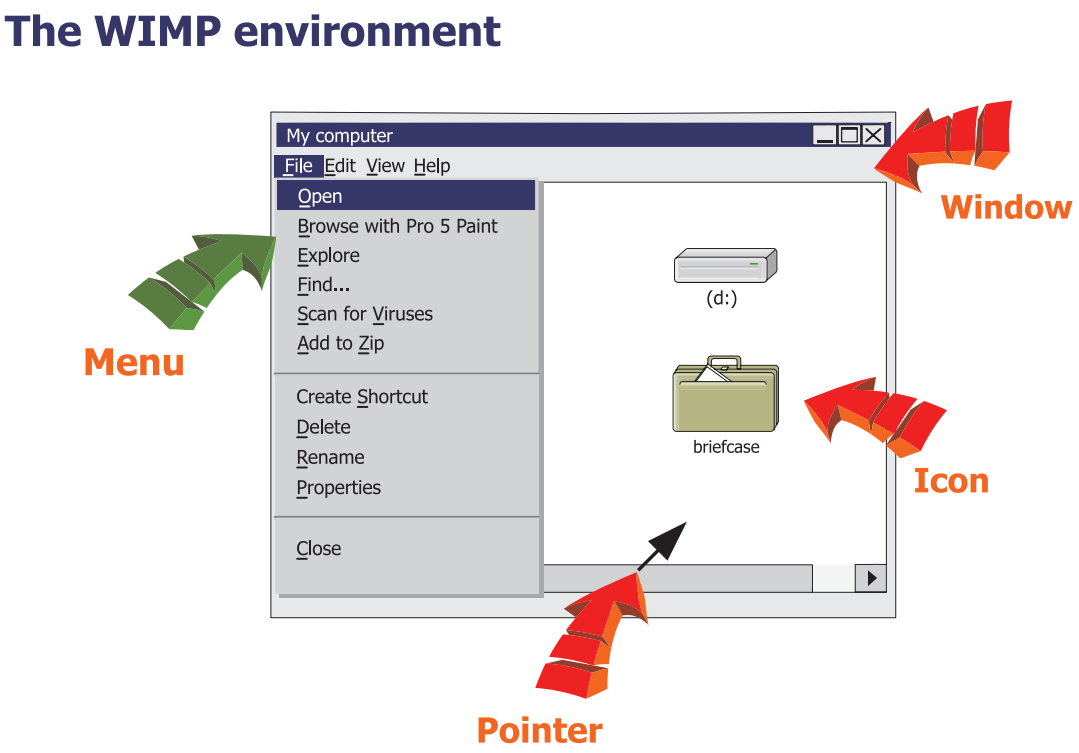
2.4.2 Designing the human-computer interface

The **human computer interface** is all that a user sees of a program.

A program's viability often relies on the quality of the HCI. A good interface makes things easier for the user. A bad interface can introduce mistakes and cause irritation.

Modern HCIs are often designed as Graphical User Interfaces (GUIs) which provide a WIMP environment. WIMP stands for Windows Icons Menus Pointers (although some

textbooks refer to Windows, Icons, Mouse, Pulldown menus). A GUI provides a set of Windows, which contain Icons and Menus. The user controls the program by means of a Pointer which reflects movement of a mouse, trackerball or other input device.



The HCI must allow easy navigation. Users should be able to move from one screen to another in a straightforward manner and to leave screens when they wish. Some sites on the Internet, for example, include a site-map to show the user how different forms are linked.

HCI must be consistent, so that similar actions in different parts of the interface have similar responses throughout the program. Prompts given to the user should be consistent. Different screens should look as though they belong to the same software package.

The HCI should provide on-line help, offering intelligible prompts and send messages and warnings to the user about the consequences of choices made, *e.g. send a warning if a user chooses to delete data*.

HCI design is based on an appreciation of what the user wishes to see. The designer thinks in terms of the windows (often called forms) that are presented to the user.

Identifying characteristics of a good user interface

On the Web is a interactivity. You should now complete this task.

2.4.3 Designing Data Structures

A program must perform operations on the data supplied to it. The data should be structured data. The choice of data structure will affect the entire program.

When the amount of data is likely to be very large, the designer must consider the

physical capacity of the hardware. For example, a million records, of a thousand bytes each, will require about a gigabyte. If the clients want these records ordered in different ways in different parts of the program, even a modern PC may have insufficient memory.

Many large systems involve a **database**. In many cases, the data is held in different tables which are linked together. These links are called relations and such a database is known as a relational database . The fields that are to be in the tables and the relations between tables need to be defined at the design stage.

Object oriented design attempts to treat data and **objects** together. An object brings together items of data and the operations that can be carried out on it. For example, a data item might be a customer's record, and the operations might include creating, displaying, editing, and deleting.

2.4.4 Other design choices

The system specification and functional specification will form the basis for designing the program. The design team must consider:

- hardware specification
- choice of high level language
- choice of operating system
- portability of the system

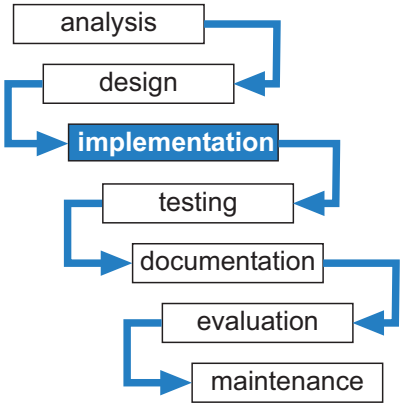
Hardware aspects will include processor speed (does the program require multiple processors for optimum performance as in networked database applications or maybe maximum memory for caching information on a regular basis). How much external storage is required for, say, regular backing up procedures and on what medium? This is an important issue especially on networked systems where users' files are archived on a daily basis.

A high level language will be chosen that is best suited to the problem but also one which the programming team is conversant with and proficient in its use. Modularity will be an important issue where the team can share the workload by compiling modules independently thereby reducing the overall development time. Module libraries can be a source of standard algorithms to be used in software projects of many types.

Choice of operating system will relate to the functionality and feel of the HCI. Windows might offer much in the way of colourful screens, interactive help and dialogue boxes etc. It must also affect how the program runs and behaves: is the OS software stable enough for the developed program to behave normally without crashing or does the OS offer a true multi-tasking environment, as in UNIX with the added benefits of in-built security. Nowadays Linux is being seen as a viable operating system which is both stable and not difficult to use.

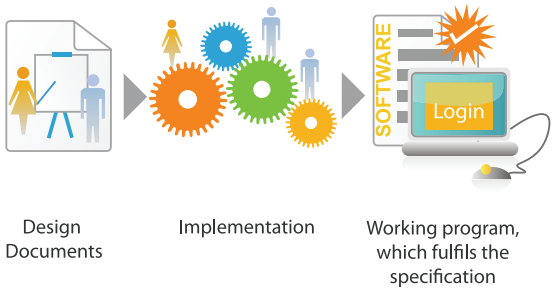
If the developed software can be moved to different machine architecture and still run to specification then it will be deemed to be portable. In some cases this might not be required but it is a characteristic that good software should possess.

2.5 The Implementation Stage



If the design has been thorough the implementation should be straightforward. It should be a matter of translating the pseudo-code into code, line by line.

The design is implemented when it has been converted into code which can be used by the computer system.



The code is written in a **high-level language**, such as Pascal, C++ or Java, and converted into code which the computer understands.

A high-level language is one that people find relatively easy to understand. The code written at this stage is called **source code**.

The machine can understand **machine code**, a translation of the source code into binary instructions. This code, because it can be executed by a computer, is also known as **executable code**.

The translation, from source code into machine code, is carried out by a program called a **compiler**. The resultant machine code is **portable** to machines of the same type running under the same kind of operating system.

Compiling and debugging large programs can take a lot of time.

Another form of translation that can reduce development time is an **interpreter**. When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter is also executing each line of code one at a time the programmer is able to see the results of the program immediately which can also help with debugging.

Sometimes problems that arise at the implementation stage will call for a return to an earlier stage of the development process. For example, it might turn out that an

algorithm runs too slowly to be useful and that the designer will have to develop a faster algorithm. Or it might be that the slowness of operation is due not to a poor algorithm but to the hardware capability. In such a case, the development team might have to return to the analysis stage and reconsider the hardware specification.

At the end of the implementation stage a structured listing is produced, complete with internal documentation. This will be checked against the design and against the original specification, to ensure that the project remains on target.

2.5.1 Debugging

At this stage, the **programming team** will make use of **test data**.

This data is designed to check that the program works properly, and that it is reliable and robust. **Testing** is often confused with the **debugging** of a program, but these are not the same, though they are very closely related.

- testing discovers any faults in a computer program
- debugging is the finding and correcting of these faults.

Maintainable software should include **internal documentation**. This is commentary within the program to explain the various stages and to record any changes that might be implemented in the coding during debugging. One aspect of this is the use of meaningful variable names.

2.5.2 Standard algorithms and module libraries

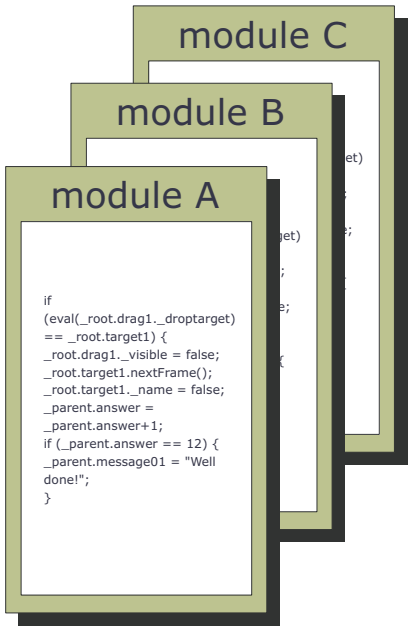
Standard algorithms

Most projects will use certain standard algorithms. Programmers need to be familiar with these common algorithms. Ones that you will become familiar with later include:

- linear searching
- counting occurrences
- finding maxima and minima

Module libraries

It is often possible to use, with or without alteration, modules that have been previously written and have been retained in a module library . A module library will include code for standard algorithms. Most development environments come with a large library of modules. Programmers can use these in the code they are developing. These libraries will include mathematical functions, modules for converting text to numbers, etc.



Each module in a module library is:

- pre-tested
- well documented

so that it can be adapted and easily used.

The use of previously written modules reduces the time spent creating the final software and minimises the cost. Designers will incorporate, if possible, modules from the module library in the design.

For many software companies and programming departments, one project will be similar to another. Projects will have component parts which are similar; for example many programs might require a sort routine to act on the contents of a database.

As time goes by, a collection of modules is built up. This collection is known as a **module library**.

Modules in the library can often be used, over and over, in new projects. This saves time spent designing and coding. If a module can be used without alteration, there is the added advantage that it has been thoroughly tested and is reliable. Use of unaltered modules reduces the time spent on the detection and correction of errors in the project as a whole.

Sometimes adjustments might be needed to a library module. After the necessary work, the module will require to undergo error and other testing.

Characteristics of module libraries

On the Web is a interactivity. You should now complete this task.

2.5.3 Review questions

Q13: Which of the following is a quality of a good human computer interface?

a) It should be Windows-based
b) It can make a program execute faster
c) It can make running a program a less irritable experience
d) It should have lots of colour

Q14: Which one of the following is not a high level language?

a) Visual Basic
b) Pascal
c) Assembler
d) C++

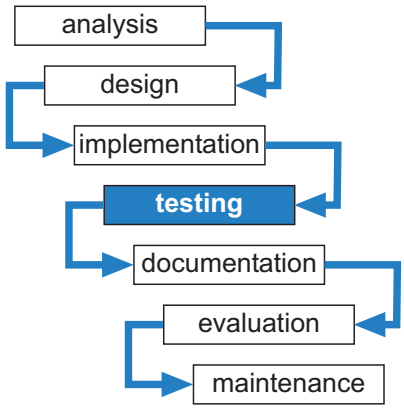
Q15: Which of these is not the result of compilation?

a) Executable code
b) Object code
c) Machine code
d) Source code

2.6 Testing

Testing has several purposes. It should check that:

- the software meets the specification
- it is robust
- it is reliable.

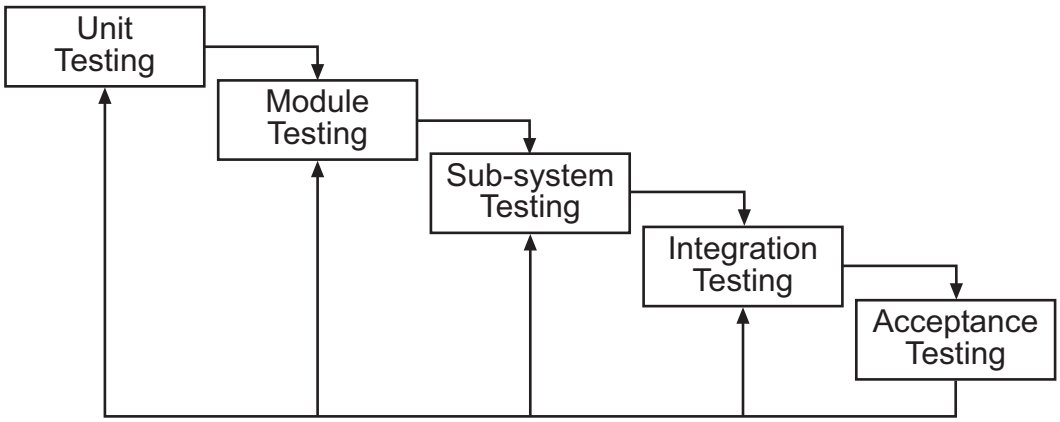


Commercial software is *not* exhaustively tested at the testing stage. Software can be complicated and the available time is limited. There must be a balance between creating a product for the market and **exhaustive testing**. If errors become apparent after release, the company will fix them and release an updated version.

Testing follows a **test plan** or strategy, involving carefully selected test data, with a view to ensuring that a reliable product has been constructed. Important aspects would be:

- what part of the program is being tested?
- what is the expected output using suitable test data?

Testing is carried out at several stages during and after implementation:



2.6.1 Test data preparation

Testing can never show that a program is correct. Even with extensive testing, it is almost certain that undetected errors exist. Testing can only demonstrate the presence of errors, it cannot demonstrate their absence.

Tests should be devised against the specification, so that you can see whether or not the program does what it is supposed to do.

In the full software development process, test data should be prepared before implementation. That is, before you have invested time and effort in writing the code.

All too often there is a temptation especially at classroom level to start coding a program as soon as possible without having produced adequate test data.

But experience shows that this is a mistake. Once you have written the code you will tend to go easy on it, and let the program's behaviour shape what you expect of it. You are going to be too kind to it.

MURPHY's LAW 1	**MURPHY's LAW 2**
The quicker program coding is started the longer the project will take	Murphy's Law 1 is correct!

Testing follows a test plan or strategy. Testing should be systematic. That means:

- testing is planned,
- everything is tested in a logical order,
- the results of testing are recorded.

With most software projects, the usual strategy is to test the software twice. The methods are called:

- **alpha testing** where the software is tested *within* the organisation
- **beta testing** where the software is tested by personnel *outside* the organisation or by certain members of the public. This is sometimes called **acceptance testing**.

2.6.2 Alpha testing

Test data is based on the specification. Data will be designed to test three aspects of the program:

- **normal operation**: data that the program has essentially been built to process; all outputs should be satisfactory.
- **extreme / boundary testing**: data to test that the program functions properly with data at the extremes of its operation; for example, if a number entered is meant to be limited, the program's performance is tested just within the limit, on the limit, and just beyond the limit; as another example, if a table is supposed to have a maximum number of elements, the program is tested to see if it can cope with exactly the maximum and if it can cope when an attempt is made to exceed the maximum.
- **exceptions testing**: data that lie beyond the extremes of the program's normal operation; these should include a selection of what might be called silly data, to test the program's robustness, that a user might enter in a moment of confusion or mischief.

Matching definitions - Testing

On the Web is a interactivity. You should now complete this task.

Faults that become evident during testing are known as **bugs**. If bugs are identified, the program is sent back, with the test logs, to the programming team for **debugging**. This process is likely to be iterative: testing, finds bugs, they get fixed, the program's tested again, more bugs are found, and so on.

It may be that bugs reveal flaws that were introduced at an earlier stage of the process, at the design or even at the analysis stage. If this is the case, the documentation for each stage of the development process will need to be corrected.

2.6.3 Beta testing

Otherwise known as acceptance testing it takes place after alpha testing. The idea is to subject a completed program to testing under actual working conditions.

If a program has been developed for use by particular clients, it is installed on their site. The clients use the program for a given period and then report back to the development team. The process might be iterative, with the development team making adjustments to the software. When the clients regard the program's operation as acceptable, the testing stage is complete.

If a program is being developed by a software house for sale to a market rather than an individual client, the developers will provide an alpha-tested version to a select group of expert users such as computer journalists and authors, and also makers of related computing products such as printers. This group is known as an independent test group. Alternatively, they may use professional software testers.

This is of benefit to both parties: the software house gets its product tested by people who are good at noticing faults, and the writers get to know about products in advance; which further benefits both parties when the final production software is released, the

software house getting publicity and the writers receive credit for being up to date.

People involved in beta testing will send back error reports to the development team. An error report is about a malfunction of the program and should contain details of the circumstances which lead to in the malfunction. These error reports are used by the development team to find and correct the problem.

2.6.4 Review questions

Q16: Which one of the following statements describe alpha testing?

a) Testing is done by the users
b) Testing is done within the organisation
c) Testing is done by specialist companies
d) Testing is done by the client

Q17: Which of the following describes beta testing?

a) The program is tested by the clients
b) The testing is more rigorous than alpha
c) The testing is for market research
d) The program is tested by specialist companies

Q18: During alpha testing, the program is usually subjected to **exceptions testing**. This means:

a) The input of unexpected data
b) The input of large numbers
c) The input of small numbers
d) All of these

2.7 The Documentation Stage

Documentation is intended to describe a system and make it more easily understood. Documentation will consist of:

1. user guide
2. technical guide

Some information may appear in both guides; e.g. system specification.

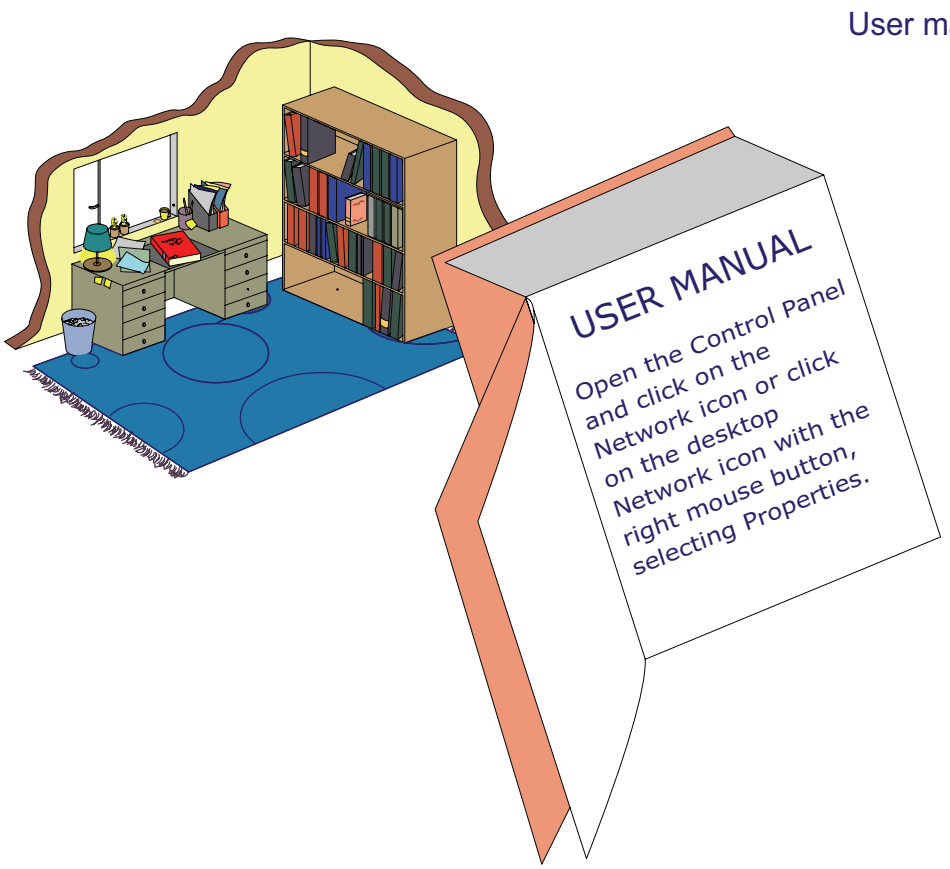
Internal documentation such as remarks or comments in the code are for the benefit of the development team. It will help if changes have to be made to the software in the future.

Other documentation for the development team includes all the documents produced in the software development process: requirements specification, program design documents (for the HCI and for the structure and logic of the underlying code), a structured listing of the code, and a test history.

2.7.1 User guide

The user guide contains information about how to install, start and use software. It should also contain a list of commands and how to use them. Where there is a significant HCI, the guide will show each form, menu, and icon, and associated instructions about their use.

User guides may be supplied as paper manuals, often with separate manuals for installation, getting started and the user guide.



Increasingly software vendors supply the manuals on disc or CD where they may be available in (pdf) or hypertext markup language (HTML).

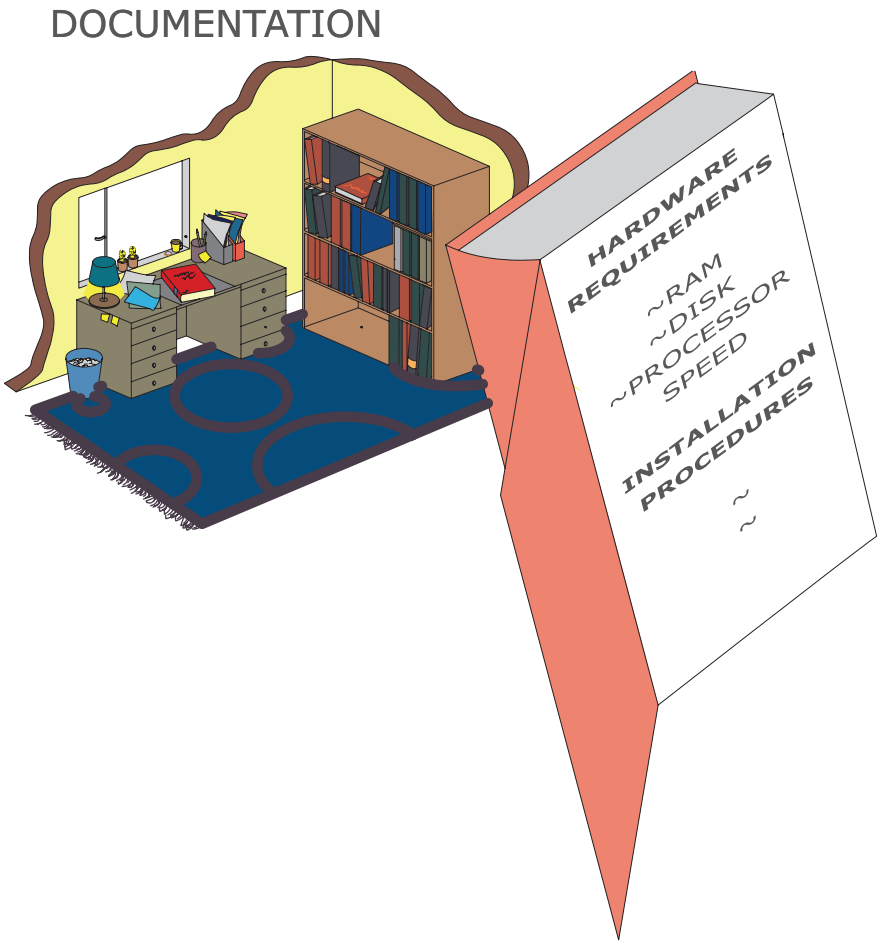
Paper manuals are costly to reproduce; manufacturers frequently include electronic files which provide up-to-date amendments. This lets the user read up to date information which could not be included in the original paper manual.

It is common for sample files to be included which complement the tutorial and provide the user with demonstration files.

The program should contain a help facility. It is common for on-line help to be presented in three tagged pages: Contents, Index, and Search. The contents present the help chapter by chapter; the index refers to certain key words in the chapters; and search offers the facility to locate key terms within the guide.

2.7.2 Technical guide

The technical guide will contain information about the hardware and software requirements of the program. The hardware specification will include details of the processor type and speed, RAM required, RAM desired, monitor resolution, graphics and sound card specifications etc. It will also contain instructions about configuring the program.



Software designed to operate, or run on networks can be very complicated and require a good deal of expertise. Technical guides can be very large and cumbersome and difficult to navigate.

Sentence completion - Documentation

On the Web is an interactivity. You should now complete this task.

2.8 Evaluation

Evaluation is the formal monitoring of a system to ensure that it is performing its purpose accurately, efficiently, cost effectively and in a timely manner. The performance of the system must be matched against a given set of *criteria* such as the initial project specification.

Evaluations of various kinds are an important aspect of the software development industry. Evaluations are used to determine if systems are usable, cost effective, conforming to performance criteria, etc. Evaluation is based on observation, interviews, and questionnaires. Additionally techniques such as automatic data logging are used. Many organisations bring in consultants who design and carry out evaluations as the skills required to carry out effective evaluations are highly specialised.

The key criterion in evaluating a software product has to be whether it is **fit for purpose** i.e. does it meet the original specification and allow the client to carry out their tasks?

The evaluation is important for the user and the software author. There are two reasons for conducting an evaluation:

- does the software meet the users' requirements?
- how can the software house improve the product?

The performance of the system can be assessed in various ways:

- how closely does it fit the system design?
- how well does it meet the problem specification?

Questions may also be asked about matters such as:

- was the project within budget?
- was the project completed to schedule?

The development team will wish to review the project, perhaps to learn from any mistakes to ensure they incorporate good points in future programmes.

Software houses aspire to produce new and better versions of their software. They will study press reviews and note any contents and criticisms. New or forthcoming changes (in technology, in operating environment, and so on) are also taken into account. When the evaluation is complete, work begins on the next version of the system.

For SQA assessment purposes, you need to be able to evaluate software in terms of:

- robustness
- reliability
- portability
- efficiency

- maintainability
  (and from Int2 level)
- readability
- fitness for purpose, and
- quality of HCI

Sentence completion - Evaluation

On the Web is an interactivity. You should now complete this task.

Evaluation terminology

On the Web is an interactivity. You should now complete this task.

2.8.1 Robustness and reliability

There is often confusion between the terms robustness and reliability.

A program is robust if it can cope with problems that come from outside and are not of its own making e.g. *corrupt input data*. Reliability is an internal matter. A program is reliable if it runs well, and is never brought to a halt by a design flaw.

When the program is complicated the distinction between the two terms is not always clear. When a machine hangs it is not always obvious whether this is due to a failure in robustness or reliability.

Robustness

The designer should try to ensure that the design is **robust**: the resulting software should be able to cope with mistakes that users might make or unexpected conditions that might occur. These should not lead to wrong results or cause the program to hang. As examples of an unexpected condition, we could take something going wrong with a printer (it jams, or it runs out of paper) or a disc drive not being available for writing, because it simply isn't there (the user's forgotten to put in the CD), or the user entering a number when asked for a letter.

Reliability

A **reliable** program always produces the expected result when given the expected input. It is designed correctly to do the task specified.

Characteristic of good software design

On the Web is an interactivity. You should now complete this task.

2.8.2 Review questions

Q19: Choose the correct response that describes the term **robust** within software development:

a) The program is strong and hardy
b) The program may be ported to a different machine architecture
c) The program can cope with mistakes that the user might make

d) The program runs to specification

Q20: Choose the correct response that describes the term **reliable** within software development:

a) The program is strong and hardy
b) The program may be ported to a different machine architecture
c) The program can cope with mistakes that the user might make
d) The program runs to specification

Q21: Using a module library can reduce software development time because (choose one):

a) Common programming modules can be used
b) Programmers do not need to write programs from scratch
c) Programmers can re-use library code in their projects
d) All of the above

2.9 Maintenance

Once the software is operating, the users will need support. In the case of a bespoke system, the development team (or the organisation it works for) may offer training in the use of the new system.

Creators of software systems often establish help desks, so users can obtain advice about the software.

Software does not wear out, in any physical sense, but the presence of errors or omissions will give rise to the need for **maintenance**.

Software maintenance always involves a change in the software with the accompanying probability that additional errors may be introduced. It is essential to ensure that adequate quality control is in place.

There are three types of software maintenance:

- corrective
- adaptive
- perfective

Corrective maintenance is concerned with errors that escaped detection during testing but which occur during actual use of the program. Users are encouraged to complete an error report, stating the inputs that seemed to provoke the problem and any messages that the program might have displayed. These error reports are invaluable to the development team, who will attempt to replicate the errors and provide a improved solution.

Adaptive maintenance is necessary when the program's environment changes. It allows the authors to provide a program which responds to changes in the operating environment. For example, a change of operating system could require changes in the program, or a new printer might call for a new printer driver to be added to the program. A change of computer system will require the program to be ported to the new system.

Perfective maintenance occurs in response to requests from the user to enhance the performance of the program. This may be due to changes in the requirements or new legislation. Such maintenance can involve revision of the entire system and can be expensive.

Matching definitions - Maintenance

On the Web is an interactivity. You should now complete this task.

2.10 Weaknesses of the software development process

Problems are usually encountered when you use this model:

1. Real projects rarely follow a linear, sequential flow. Apart from any software problems, people change their minds, and often there may be changes in legislation which mean that the program must be altered in order to comply with the new regulations. No matter what the reason, iteration always occurs and this creates problems because much of your work has to be re-examined and revised;
2. It is difficult for the customer to state all requirements explicitly at the start of developments. The model depends on this and has difficulty incorporating customer uncertainty;
3. Clients are frequently excluded from the development. The working version of the program will not be available for the customer to see until late in the development cycle;

4. Developers work in isolation from the clients, often for months, only for the clients to be disappointed with the results. Many developers value feedback from clients as the project progresses;

5. Errors arising from incorrect requirements will not be obvious until late in the cycle, by which time they will be difficult and expensive to fix. There is nothing so depressing as delivering months, sometimes years, of work to the customer only to be greeted with the response, "That's not what I wanted at all."

2.11 Summary

The following summary points are related to the learning objectives in the topic introduction:

- the software development process has 7 stages (analysis, design, implementation, testing, documentation, evaluation, maintenance);
- the process is iterative and involves a continual revision and evaluation at each phase of the process.

2.12 End of topic test

An online assessment is provided to help you review this topic.

Topic 3

Tools and techniques

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe and use pseudocode;*
- *describe and use a graphical design notation (structure diagram or other suitable method);*

Learning Objectives

After studying this topic, you should:

- *understand the nature of graphical design notations*
- *be able to use a graphical design notation*
- *understand the nature of pseudocode*
- *be able to use pseudocode*
- *understand the nature of top down design and stepwise refinement*

- *be able to describe the main types of error*
- *be able to explain how good test data can reduce errors*

Revision

Q1: At the design stage of the software development process pseudocode may be used to represent a solution to a problem. Which of the following best describes pseudocode?

a) It uses ordinary English words
b) It is high level language dependant
c) It is very useful in complex program designs
d) It mostly uses high level language key words

Q2: A structure diagram is a valuable aid to the programming team. This is because:

a) They are easy to use and can be understood by the user
b) They allow for faster program execution
c) They are required during evaluation
d) They represent the design in a visual way

Q3: Pseudocode can be considered to be an intermediate stage between:

a) High level language code and machine code
b) English and high level language code
c) English and machine code
d) Source code and object code

3.1 Introduction

In this topic you will learn about the various methods that are used to aid the system developers and programming team to implement solutions according to program specifications.

The first important issue you will come across is that program coding is only attempted after extensive and rigorous series of analysis and design stages are completed. Efforts at these initial stages pay dividends at the coding stage reducing testing, debugging and maintenance of the programs.

The main tools and techniques include:

- Design methodologies
- Test data
- Structured program listing
- Comprehensive error reporting
- Module libraries

3.2 Design methodologies and notations

The design of software is something of an art and normally follows a clear design **methodology**. A methodology is an agreed technique used to design software. It includes both approaches to designing software and the notations used to represent the design.

Design documentation will often include more than one notation, e.g. **structure charts** for the higher levels, to provide an overall picture, and the pseudo-code for the detail. Often the design notations are used at different stages in the design process and serve to provide information to different audiences.

3.2.1 Top-down design with stepwise refinement

Top-down design with stepwise refinement is an approach used in computing and in many other fields as well. The idea behind it is this:

1. A problem might be difficult to solve as it stands, so you try to break it down into a set of smaller problems which might be solved more easily. This represents the first step of the process
2. You then take the smaller problems, one at a time, and break them down into still smaller problems
3. You keep repeating this process of breaking the problems down until all the problems facing you are small enough to handle

At this point you are able to create detailed designs, which can be turned into programming code.

The process is top-down, because you start with an overview of the whole problem and gradually work down to the fine detail. It is called 'top-down with stepwise refinement' because the designer works through a series of steps, gradually refining the small detailed aspects of the program.

One other advantage of this systematic approach is that it also automatically gives a structure to your solution.

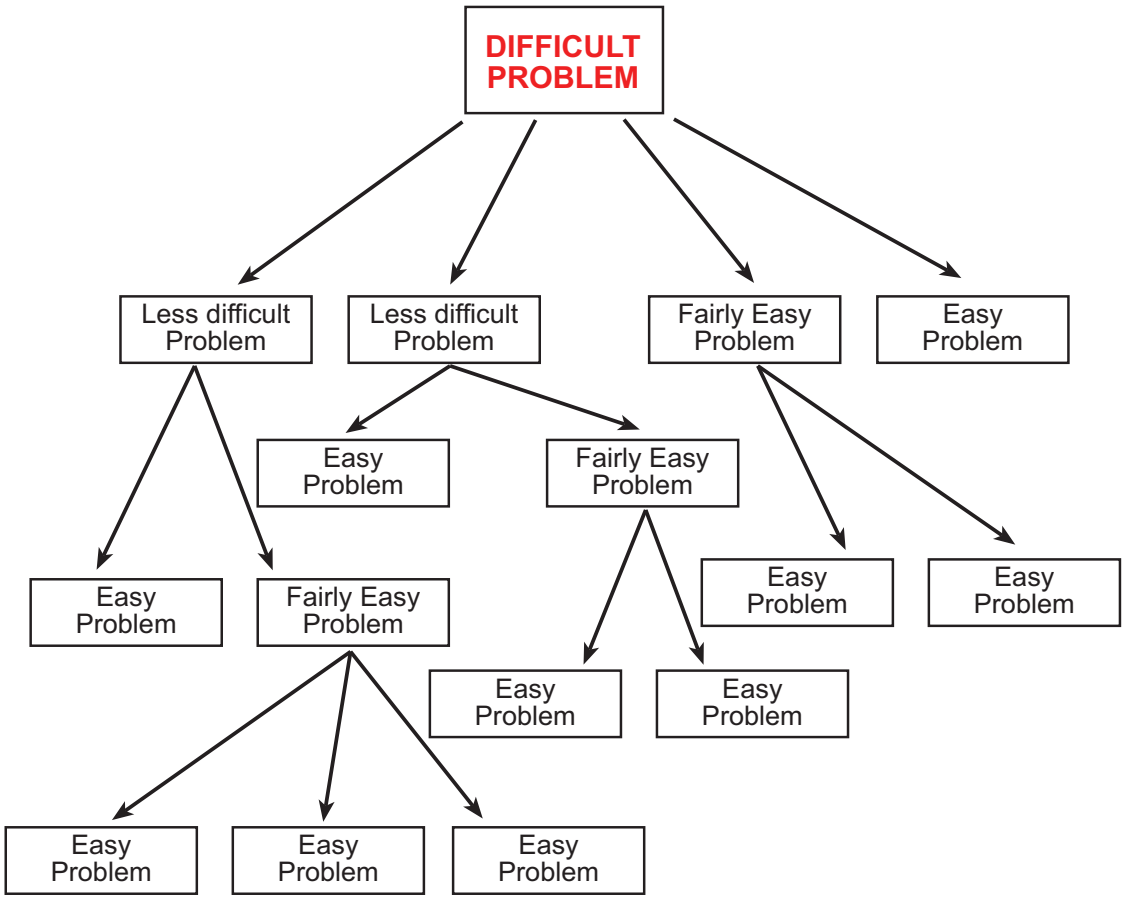


Figure 3.1: Stepwise refinement

Stepwise refinement allows the designer to concentrate on one small part of the problem at a time. If thinking about the problem as a whole, the designer does not have to bother with details. In thinking about a portion of the detailed design, the designer does not have to bear in mind all the rest of the problem. This makes the process manageable.

Once manageable parts have been identified the analyst can assign individual tasks to different teams of programmers. The complex task is made more straightforward by the system of *'divide and conquer'*.

Programmers often use **stubs** when using a top down technique. A stub is an outline of a module, that does little more, at run time, than declare its presence or return a value of appropriate type. When all the stubs are in place, the program as it stands can be tested to make sure that all the stubs are properly linked. Then the detailed work can begin.

Advantages of top-down design

On the Web is an interactivity. You should now complete this task.

3.2.2 Structure charts/diagrams

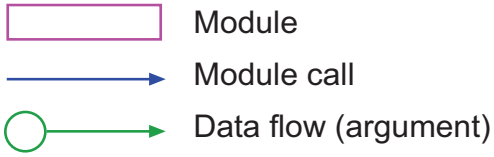
An important approach in practically all methods of analysis is to draw a picture of some kind. Analysts use diagrams which show:

- the main elements of the solution,
- how they fit together.

It is not enough for the analyst to have a design in mind. It must be represented in a form that can be used by all members of the team. One method is to use structure charts, which are sometimes called structure diagrams. A structure chart gives a picture of the design. A structure chart is an example of a **graphical design notation**

Styles vary in detail, but in essence structure charts consist of boxes, lines, arrows and text. A box represents a block of code and has a name. Usually, it is a descriptive expression starting with a verb; it shows the block's purpose.

Structure charts can be drawn according to various sets of guidelines. These are the basic graphical elements:



A structure chart for a program that simply gets in some data, changes it in some way and produces output might look like Figure 3.2:

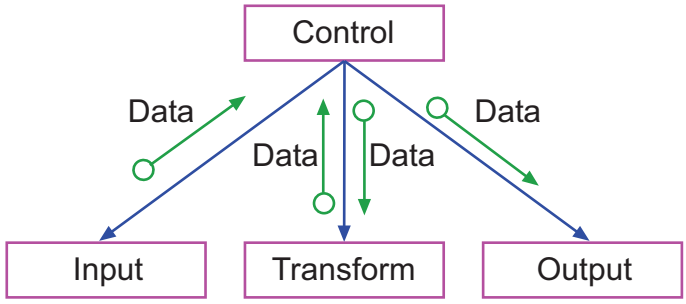


Figure 3.2: Software Design Structure

3.2.3 Interpreting Structure Charts

Structure charts are read from top to bottom and left to right at each level. The line joining two boxes indicates that the lower is called, or brought into action, by the upper. The lower box represents one of the things the upper box has to do. Along each line, names and arrows describe the data that flow from one block to another and the direction in which they flow.

Each block in the structure chart represents a section of code to be written. These sections are called modules.

The modules in a structure chart will become modules of code in the finished program. The designer must give the modules meaningful names (rather than Block 1, Block 2

etc.) which describe what the module does. This makes the design easier to follow.

The project as a whole is at the top. A structure chart shows the relationship between modules and in particular shows which modules contain calls to modules lower down in the structure. A complete structure chart of a large software system shows the relationship between all the modules in the system.

By convention, structure charts are kept simple, with no more than half a dozen blocks to each chart. What matters is that the picture should be clear and easy to bear in mind. If a block needs further refinement, that can be represented in another chart.

Describing a structure chart

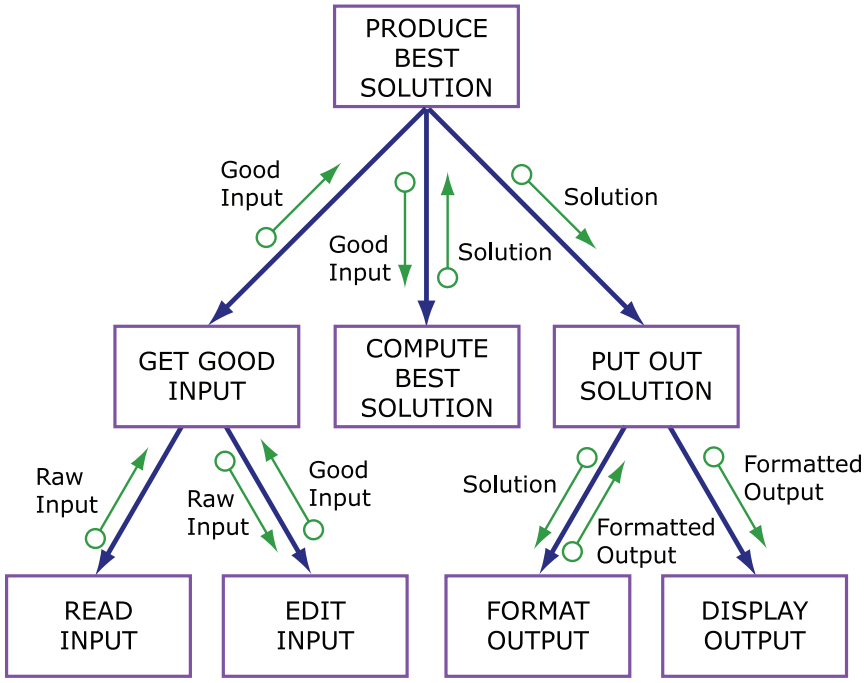
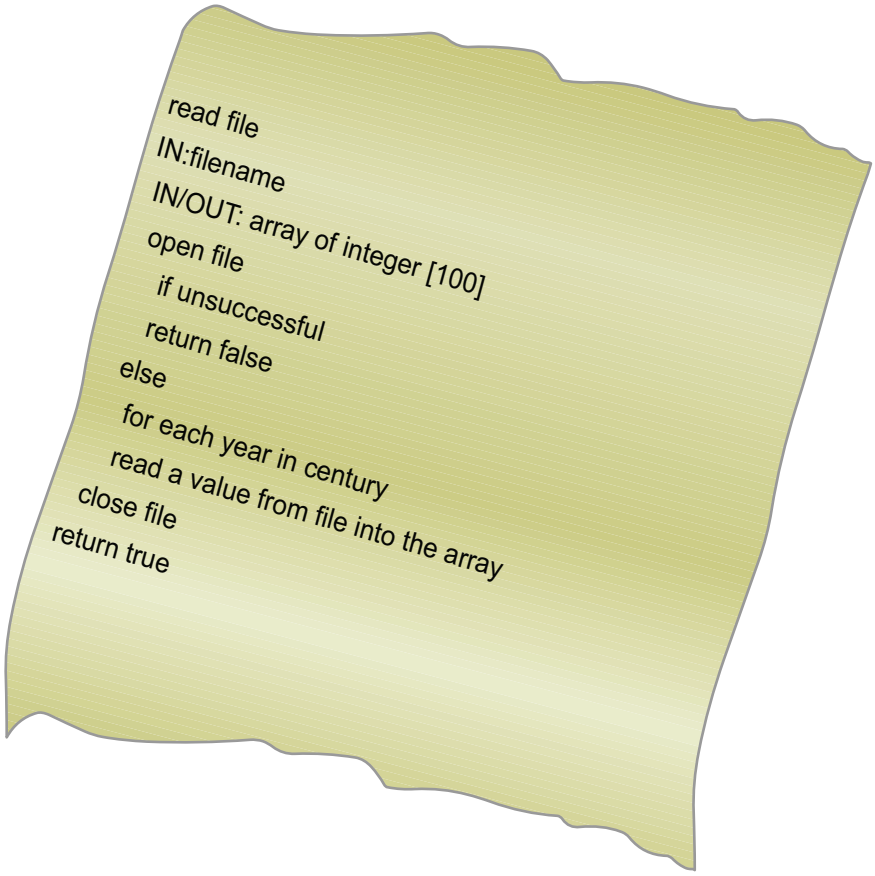


Figure 3.3: Sample structure chart

Figure 3.3 shows a structure chart. Make a list of all the modules and all the data flows. Remember to make a sensible guess at what each block is supposed to do - its name should be a good indication of this!
What sort of program do you think is represented by the whole structure chart?

3.2.4 Pseudocode

Another method is to describe the design in terms of pseudocode. Pseudocode is a **non-graphical design notation**. This is mostly used for working out the details of a design. At this level, pictures don't really help. Pseudocode is a way of writing that lies somewhere between code and natural language (such as English). It represents an understanding of the solution that can be turned into code but can still be understood by people. Pseudocode follows the indenting conventions of the programming language to be used for the project. This indentation makes the pseudocode easier to follow and to understand.



Pseudocode is a way of writing about a process without having to bother about details which are simply a matter of coding. So that you could write:

```
convert entry to number
```

without having to remember exactly how this is done in the chosen programming language.

Pseudocode frees us to get on with thinking about the details of program design, without our having to stop and look things up in books or worry about whether we have got our syntax exactly right. You use pseudocode to explain what a process will do in a clear and concise way. One of the advantages of pseudocode is that it can be translated into programming language code fairly easily. By writing out your program in this way, you are opening up the possibility of re-writing it in a large number of different forms, suitable for different machines.

Many designers use a numbering system for parts of the design. Each block in the structure chart has its identifying number and each line of pseudocode is also numbered. These numbers enable parts of the program to be related, and show their dependencies. For example, a block number 3.1 in a structure chart might indicate that this is the first sub-module in the third module in the main part of the program. A line of pseudocode 3.1.10 would indicate the tenth line in this module.

Here is an example of a standard algorithm that you will meet in your programming exercises, written in numbered pseudocode:

Counting occurrences

```
1.      set counter to zero
2.      prompt user to input search value
3.      set pointer to start of list
4.      do
4.1       compare search value to item at current list position
4.1.1       if search value = current item then
4.1.2       increment counter by 1
4.1.3       set pointer to next position in the list
4.2       loop until end of list
5.      output number of occurrences(counter)
```

Characteristics of pseudocode

On the Web is a interactivity. You should now complete this task.

3.2.5 Review questions

Q4: Which of the following is a graphical design notation?

a) Structure charts
b) Stubs
c) Pseudocode
d) Stepwise refinement

Q5: What statement refers to the term **top-down design**?

a) It is a software testing approach
b) It is breaking complex problems down into smaller units
c) It is the detailed design of software logic
d) It is the use of pseudocode

Q6: Which of the following is a design methodology?

a) Structure chart
b) Pseudocode
c) Stepwise refinement
d) Dry running

Q7: For a designer, an advantage of pseudocode is that he/she can?

a) Think more about the solution to the problem
b) Think more about the hardware
c) Think more about how the program will run
d) Think more about the speed of execution of the program

Q8: One advantage of using stepwise refinement at the design stage is that it:

a) Reduces the chance of errors being introduced
b) The designer does not have to bother with too much detail
c) Makes the process more complex to novices
d) The designer can concentrate on a small part of the problem at a time

3.3 Test Data

The purpose of test data is to determine that the system behaves as expected and is correct according to the program specification.

Preparing test data

A program is to be written which will read a list of examination marks typed in at the keyboard and find the average examination mark. The program should reject examination marks which are not within a specified range and provide a suitable error message to the user.

Q9:

1. Write down what you think an acceptable range for an examination mark would be;
2. Write down an algorithm, in pseudocode notation, to represent the data input part of a solution to this problem;
3. Copy the headings in the table below and construct at least 8 test data entries to demonstrate how you would verify the correctness of the program?

Test Case	Reason	Expected Result	Actual Result	Comments

Sentence completion - test data

On the Web is an interactivity. You should now complete this task.

3.3.1 Review questions

Q10: Which one of the following options states the main purpose of test data?

a) To determine that the system meets the specification
b) To prove the absence of errors
c) To show that the programmers have been careless
d) To minimise the number of errors in the program

Q11: Error detection can be time-consuming. Which one of the following could be added to a program to help detect errors?

a) Suitable commentary throughout the program
b) Output statements at key points in the code
c) Specific error-detecting code
d) Nothing can be added

Q12: One example of a fault-avoidance technique in developing software is?

a) Take more time to design better software
b) Hire more programmers so that errors will be easier to detect
c) Design input routines that will not crash when presented with unexpected data
d) Make sure that that the code is specifically written to avoid errors

Q13: The last remaining errors in a program are not easy to remove because

a) They could remain hidden until the program is run under all conditions
b) The compiler is not very efficient
c) De-bugging procedures are not effective enough
d) They will not affect the running of the program since most errors have been found

3.4 Structured Listing

A structured listing is a hard copy of the program source code.

It is important that the source code is laid out in accordance with the conventions of the implementation language.

- The code should be properly indented; this helps people follow the structure of the code.
- Meaningful names should be used for modules, constants, and variables.
- The use of **internal commentary** will help to explain the logic of the code to others and also serve as a documentation aid for the programmer.

A structured listing can be produced at any time during implementation. It can serve as a tool for checking program logic and also form part of the final software documentation.

Figure 3.4 shows a structured listing of a program coded in the language Visual Basic.

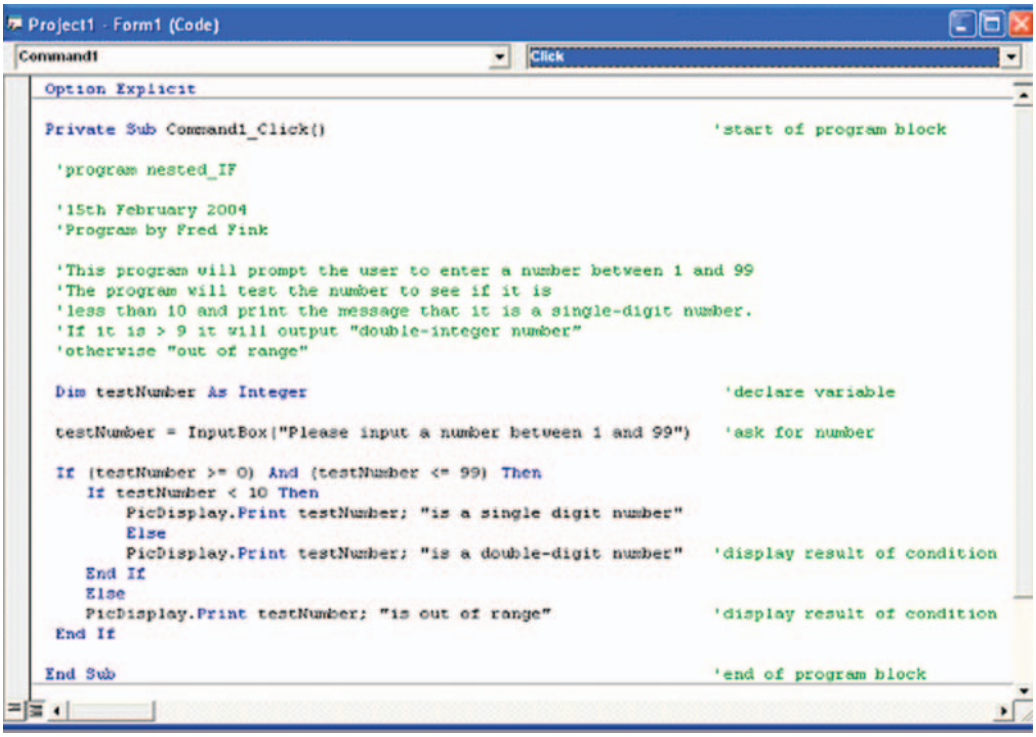


Figure 3.4:

3.5 Error Reporting

Errors can occur at any of the stages in the development process.

This section will describe the following 3 categories of error:

- syntax errors
- run-time errors
- logical errors

Syntax Errors

These are errors which result from incorrect use of the programming language structure. They are detected during compilation and examples include:

- use of programming language keywords as variable identifiers,
- mis-spelling of keywords, e.g. `PRNIT` instead of `PRINT`
- blocks that have a missing "end" marker - e.g. `FOR` without `NEXT`
- missing brackets/semi-colons.

These types of error can be very irritating, as they prevent the program from compiling. The program cannot run.

Run-time Errors

These are errors detected during the execution of the program. Even given that a program is translated into machine code and runs on the computer, there is no guarantee that it will not generate errors. For example, a common mistake made by programmers is to divide by zero. In most cases this will produce a run-time error and cause the program to halt unexpectedly.

Another example includes attempting to read character data directly as numeric.

'Array Bounds Exceeded' is a common mistake in languages which support array data structures. Here, the programmer is attempting to access a position within the array which exceeds its predefined bounds.

Logical Errors

These are errors in the **design** of the program; *e.g. calling the wrong procedure or routine.*

Even given that the program translates to machine code and does not halt unexpectedly due to run-time errors, there is no guarantee that it will not fail. It may contain logical errors. These are errors in the logic of the code itself. For example, writing code to add two numbers instead of multiplying them, or forgetting to write code to do something when a condition that is being tested fails.

Other examples include:

- making a call to the wrong module
- passing incorrect data into a module
- passing the wrong data out of a module.

Error types

Online there is an interactivity which you should attempt.

3.6 Summary

The following summary points are related to the learning objectives in the topic introduction:

- much has to be done in terms of design and testing of software before the coding and implementation stages;
- various graphical design constructs are available, including structure charts and diagrams;
- top-down design with stepwise refinement is a well-established technique;
- testing is an extremely time-consuming process and should find and resolve all syntax errors, run-time errors and logical errors;
- the main aim is to produce software that is reliable, robust, portable, efficient and maintainable;

3.7 End of topic test

An online assessment is provided to help you review this topic.

Topic 4

Personnel

Contents

Prerequisite knowledge

Before studying this topic, you should be able to:

- *describe the 7 stages of the software development process.*

Learning Objectives

- *Identify the personnel at each stage of the Software Development Process*
- *Understand the role of each person*

4.1 Introduction

The Software Development Process involves many people throughout a computer system's lifecycle. In this topic you will meet the personnel involved, with a brief outline of their activities.

Although many people may be involved, the key personnel are discussed under the five main headings:

1. client
2. project manager
3. systems analyst
4. programmers
5. independent testing group

In a nutshell the process encompasses the following events:

1. The company who require the new or updated system are the clients. They approach external consultants, the people who will create the system.
2. The consultants appoint a **project manager**, who carries out a feasibility study. If the feasibility study bodes well, the management asks for a full system investigation. This is carried out by a systems analyst from the consultants, who works with the project manager. It culminates in an operational requirements specification.
3. When the operational requirements have been agreed by both the consultants and the customer, a contract for the system is drawn up.
4. The Systems Analyst develops a design for the solution.
5. The consultants put a team of programmers on the job to code the design, and the software development process begins.
6. Once the software has been implemented in a chosen language it undergoes rigorous testing by an **independent test group**.

Identifying Personnel involved in a Systems Development

On the Web is an interactivity. You should now complete this task.

4.2 The Client

In a software development project, the client is the person or organisation which requires the new software to be developed. The client might be:

- a major multinational company
- a government department
- a small company
- a charity
- an individual

It is important to note that the terms client and user do not necessarily mean the same thing, but are sometimes used interchangeably.

A client is someone or a group such as management who **buys** or intends to buy some software for a particular purpose.

A user is someone who **uses** or will use the software.

The appropriate personnel within the client organisation will be concerned with the following activities:

- holding discussions with the consultants
- reviewing reports, and negotiating contracts
- providing relevant information to the project manager and other members of the consultancy team
- deciding whether or not to go ahead with the project
- paying the consultants

Sentence completion - software development

On the Web is an interactivity. You should now complete this task.

4.3 The Project Manager

The project manager is responsible for the project. The project manager will be appointed by the consultants.

The project manager supervises the project and carries out the initial stages. If it goes ahead, the project manager will take charge of the project, from the first brainstorming session to the software launch and implementation.

It's up to the project manager to keep the process on schedule by whatever means possible. Sometimes it may involve 'cracking the whip' to make sure the work is up to standard and that important deadlines are met - being a project manager means being accountable for the entire duration of the project.

Project managers are easy targets for criticism because the success or failure of a project is often dependent on their decisions. Hence it can be a thankless task at times for this overworked person!

The most time-consuming undertaking of all will be managing people whether they be clients or as members of the consultancy team.

With clients, the project manager:

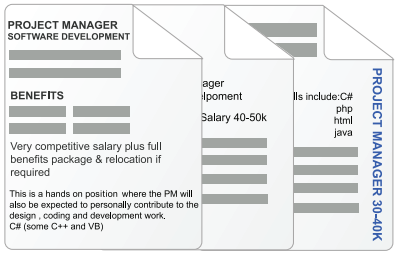
- needs to extract all the relevant information from the client and force him/her to make decisions regarding software specification

With his/her own team, the project manager:

- promotes good welfare by raising team spirit through regular, personal meetings and positive appraisal.

A job description for a typical project manager might involve the following requirements:-

- have experience in dealing with a variety of clients
- the ability to manage a team
- possess excellent verbal and written communication skills
- have strong attention to detail
- possess the ability to create schedules and budgets
- show remarkable endurance under stress



4.4 The Systems Analyst

The systems analyst carries out the system investigation. (In large systems, the analysis might be carried out by a team of analysts.)

The systems analyst is appointed by the project manager.

Systems analysts are active in the analysis, design, testing and implementation phases of the project.

Systems analysts will usually have some programming experience but they won't necessarily be programmers.

The aim of the systems analyst is to produce a clear specification that the rest of the development team will use in the subsequent stages.

Analysts begin an assignment by discussing the systems requirements with the client

and potential users to determine the exact nature of the problem. They define the goals of the system and divide the solutions into individual steps and separate procedures.

Full systems analysis has three phases:

1. collection of information
2. analysis of information collected
3. production of a problem specification or user requirements specification.

Systems Analyst Research

Use the internet to find 4 adverts for systems analyst jobs, and use the information to a complete the following table:

job title	salary	skills / experience required

4.4.1 Collection of Information

There are various methods used for gathering information. These include:

(a) Interviews.

The analyst talks face to face with clients, to find out how the current system works and what is required of the new system.

The analyst seeks answers to the obvious questions beginning with these words:

- *what* does the system do?
- *where* are these things done?
- *when* are they done?
- *why* are they done?
- and *who* does them?

Many of these techniques are essentially iterative. The answers to questions often raise further questions, which the analyst must go and ask, and so on.

(b) Questionnaires.

Where the current system has a large number of users, the analyst might construct a questionnaire for everyone involved. People often respond more frankly to an anonymous questionnaire. On the other hand, the response rate can be low.

(c) Observation.

The analyst studies the current system and observes how it works. This is useful for bringing to light things that users take for granted.

(d) Document analysis.

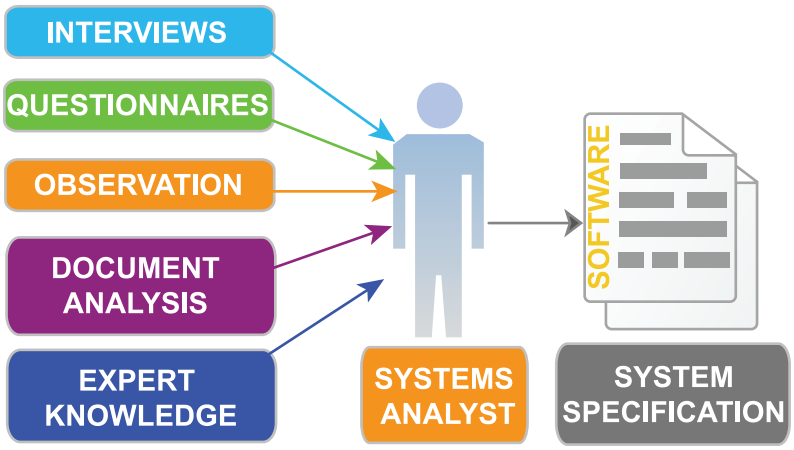
Many different kinds of document are involved in a system: the documents that the system produces; the documents it uses i.e source documents and the documents that affect how the system works (such as documents that spell out the procedures to be followed in using the system). The first gives the analyst an idea of what the new system will have to produce, and the second will help understanding of the workings of the current system.

(e) Expert Knowledge

The analyst has to try to gain understanding of the processes the new system is supposed to help.

This understanding is important because particular people tend to take their own special knowledge for granted and as a result don't mention important requirements to the analyst.

A basic understanding of the process will allow the analyst to ask more searching questions during the interviews with key informants.



4.4.2 Production of the problem specification

Once the analyst has arrived at a clear idea of the problem, this is expressed in a document called a specification. It may be called:

- the problem specification,
- requirements specification,
- the system specification, or
- software specification.

The specification includes a full description of the problem. All the inputs, processes, and outputs are described. No system works on its own, independently of the outside world: certain assumptions have to be made about the boundaries between a system and its environment, and these must be indicated, described and included.

The specification represents a legal agreement between the clients and the development team. The process of reaching agreement will be iterative: the analyst will present a draft specification to the clients, who will suggest amendments, and so on, until a specification is agreed.

The specification is used throughout the rest of the development process. Material produced is based on it and compared with it. For example, test data will be drawn up on the basis of the specification.

The specification can be used as a checklist, to ensure that the development process remains on target.

4.4.3 Review Questions

Q1: In the development process the client is best described as (choose one):

a) The group who will use the software
b) The group who will test the software
c) The group who will purchase the software
d) The group who will write the software

Q2: In a company the idea of starting a new project is mainly to:

a) Make as large a profit as possible
b) Use up surplus capital otherwise it will diverted elsewhere
c) Give the users in the company something else to do
d) Benefit the organisation in some way

Q3: Which one of the following statements is NOT true of the systems analyst?

a) The systems analyst is appointed by the project manager
b) The systems analyst discusses the system requirements with the clients
c) The systems analyst determines the exact nature of the problem
d) The systems analyst is responsible for the entire project

Q4: The purpose of the systems analysis is to:

a) Allow the systems analyst to produce a clear specification of the problem
b) Allow the client to say how much the project will cost
c) Allow the project manager to dictate whether the project should go ahead
d) Allow the collection of information that the company will need anyway

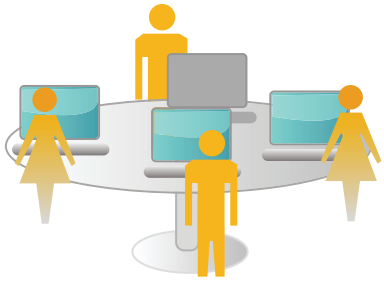
Q5: If a project falls behind schedule, who is responsible for getting it back on track:

a) The client
b) The project manager
c) The systems analyst
d) The project will eventually recover itself

4.5 The Programming Team

The programming team is responsible for the implementation phase of the software development process. They work to the design created by the systems analyst. They are also responsible for testing the software, and for maintaining it once it has been installed. The team will report to the systems analyst.

The analyst schedules the work, keeps an eye on performance, and oversees the development of the system.



In respect of production, a programmer's work is in two parts:

First the detailed logic of the modules in the system has to be worked out. Data flow has to be identified at all stages.

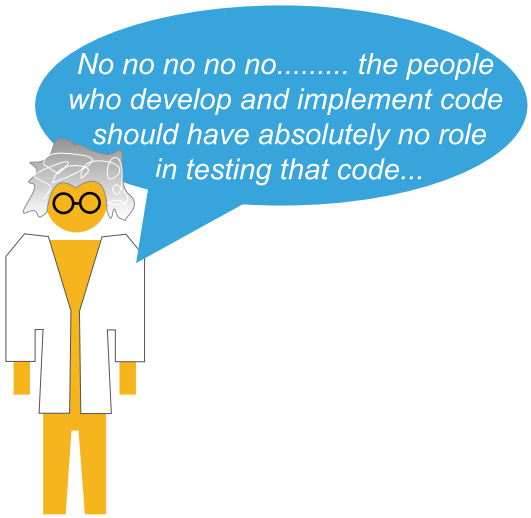
Second, the programmers write the code, test it and debug it though testing is often done by a separate team.

In most cases, several programmers work together as a team under a senior programmer's supervision.

Programmers test a program by running it, to ensure the instructions are correct and it produces the desired information. If errors do occur, the programmer must make the appropriate change and recheck the program until it produces the correct results. This process is called debugging. Programmers may continue to fix these problems throughout the life of a program.

4.6 Independent Test Group

Consider the following quote from a computing expert in the field of software development:



Quite an adamant view with a clear message!

Programmers would be less inclined to test their software to destruction and take into effect test data that might cause unexpected results. - they have the knowledge of how the functionality of the code was designed and will test accordingly. Also they probably don't have sufficient time to fully test the program under all conditions.

The software is, therefore, passed to other groups of people - **independent test groups** who will undertake impartial testing.

Roles within Software Development

On the Web is an interactivity. You should now complete this task.

4.6.1 Roles in classroom programming

In real-life programming, all these roles are carried out by different people.

When it comes to the work you do in the classroom:

- the SQA or your teacher are the client
- you are the project manager, systems analyst, programming team and tester!

4.6.2 Review Questions

Q6: The program specification is an important document in the software development process. The main reasons for this is:

a) It can be used as a checklist to ensure the project is on target
b) Test data will be drawn up on the basis of this document
c) It describes all the inputs, outputs and processes involved in the project
d) All of the above

Q7: Which one of the following is not true of the programming team?

a) They report directly to the project manager at all stages of programming
b) They are able to modify and repair existing programs
c) They are responsible for all test strategies concerning the software
d) They are overseen by the systems analyst at all stages of their work

Q8: The completed project is usually tested by an independent group. This is because:

a) Independent test group will have better facilities for testing software
b) Programmers will tend to test only within the functionality of their own code
c) Independent test group will not test further than they have to so saving time
d) Programmers are happy to test their programs to destruction

Q9: If a project begins to run over budget, who is responsible for dealing with the problem?

a) The client
b) The programming team
c) The systems analyst
d) The project manager

Q10: In beta testing, which one of the following is true?

a) Testing is done in-house
b) Testing is done by specialist personnel at cost
c) Testing is done by external groups on a variety of computer platforms
d) Testing focuses on the problem specification

Online there is an interactivity "Matching personnel to stages" which you should attempt now.

4.7 Summary

The following summary points are related to the learning objectives in the topic introduction:

- many different people are involved in the software development process;
- the client (who may be an individual or an organisation) is the buyer of the software being developed;
- the users are the people who will actually use the software;
- the project manager takes overall responsibility for the whole project;
- the systems analyst clarifies what the client requires, draws up a specification and produces a design;
- the programming team implements this design, and may carry out testing;
- an independent test group may be used to carry out systematic and comprehensive testing.

4.8 End of topic test

An online assessment is provided to help you review this topic.

Topic 5

Languages and Environments

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe and compare machine code and high level languages;*
- *explain the need for translation of high level language;*
- *describe the function of a compiler;*
- *describe the function of an interpreter;*
- *describe the process of recording a macro;*

- *assign a keystroke to a macro;*
- *describe examples of the use of macros;*
- *describe the features of a text editor.*

Learning Objectives

After completing this topic, you should be able to:

- *understand the differences between procedural, declarative and event-driven languages;*
- *describe the uses of compilers and interpreters;*
- *compare the functions and efficiencies of compilers and interpreters;*
- *describe the features and uses of scripting languages;*
- *describe how to create and edit a macro;*
- *describe the need for and benefits of scripting languages.*

Revision

Q1: Programs are written in high level languages and then translated before they can be executed by a computer. This is because:

a) computers do not understand English words
b) it is easier to write high level code
c) computers only understand machine code
d) all of the above

Q2: Two translator programs are a compiler and an interpreter. Which one of the following statements is true?

a) A compiler produces object code translating a program line by line
b) A compiler produces object code for a whole program in one operation
c) An interpreter produces object code for a whole program in one operation
d) A compiler translates and executes a program line by line.

Q3: Which one of the following error situations would be picked up by either a compiler or interpreter?

a) Run time error
b) Logic error
c) System error
d) Syntax error

5.1 Introduction

This topic will introduce you to the various types of programming languages that would be available at the implementation phase of the software development process. As you will see the choices are numerous and expertise is required in matching the system specification with the most appropriate programming language available. How programming languages translate source code into object code is further explained, emphasising the advantages and disadvantages of the methods used.

5.2 Programming Languages

During the implementation phase of the software development process the program design will be coded using a suitable high level programming language. High-level languages help software developers to identify more with the problem rather than the hardware on which the final program will run i.e. they are **problem oriented**. Program statements and expressions in such languages generally incorporate English words which mean that they are easier to understand and use than machine code.

There are many programming languages that could be used for the implementation phase of software development, such as C and C++, Pascal, and Java. Older languages still much used include COBOL and Fortran. COBOL is used mostly on mainframes for batch processing: organisations still use it for new programs because all their old

programs are written in it. Fortran is still used for applications where the emphasis is on numerical processing (such as processing meteorological data.)

In this topic and others, all language features are exemplified using Visual BASIC

The choice of language should be based on:

- which language has the facilities most appropriate to solving the required problem
- a suitable compiler/interpreter available for the client hardware.

5.3 Classification of High Level Languages

At the time of writing it has been estimated that around 2,500 programming languages have been catalogued over the years.

Programming languages

For further information and a chance to download a time line of language development as a poster, consult the following links:

http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

http://www.levenez.com/lang/history.html#08

Look up any computer languages you have used.
Q1. When was it "invented"?
Q2. How many versions have there been since then?
Q3. What did it develop out of?

Classification of programming languages is fraught with problems since, as you will see, many languages can fall into more than one category.

Historically programming languages were classified according to whether they were

- a **general purpose language** that could be applied to a broad range of situations,
- or a **special purpose language** that were designed for specific tasks.

Table 5.1 summarises some high level languages and variants:

Table 5.1: Historical classification of programming languages

Area	**Language(s)**	**Purpose**
General purpose	ALGOL , ALGOL 60, ALGOL 68 *(ALGOrthimic Language)*	Coding of general algorithms. ALGOL 60 was the first defined language i.e. code produced identical results on any mainframe computer world-wide. Also introduced procedural programming and parameter passing. Derivations are: Pascal, C, C++ and COMAL
Scientific	FORTRAN *(FORmula TRANslation)*	Developed in the 1950s for use in scientific and engineering applications and is still in use today.
Commercial	COBOL*(COmmon Business Oriented Language)*	Suitable for data processing applications. So widely used, COBOL 2002 is the latest version in use.
Education	Pascal	Developed from ALGOL, developed in the 1970s to teach structured programming
	BASIC (Beginners All Symbolic Instruction Code)	One of the first interpreted languages, designed for beginners to programming in the 1960s.
	COMAL (COMmon Algorithmic Language)	The language promoted the use of structured code as opposed to the 'spaghetti code' from using BASIC.
Artificial Intelligence	PROLOG*(PROgramming in LOGic)*	Used in the construction of AI applications, expert systems and in the teaching of AI.
Operating systems	C	Derived from UNIX as the 'C' shell it now has many uses in programming, being very close to assembly language.

Alternatively high level languages can be classified according to their structure and purpose. Although quite an extensive list, for the purposes of this topic the following categories are of importance:

1. Imperative/procedural
2. Declarative/logical
3. Event-driven
4. Scripting

   Two other types (which you do not need to know about for this course) are:

5. Object-oriented
6. Functional

A table based on this language classification is shown Table 5.2

Table 5.2: Classification based on structure

Programming Language	**Structure**	**Purpose**
Pascal	procedural	general purpose language, widely used
Visual BASIC	event-driven	windows interface applications, multimedia
PROLOG	declarative	artificial intelligence
Visual C++	event-driven	used as front end to develop user interface
COBOL	procedural	business use
Java	object-oriented	platform independent - an object oriented language
VBscript	scripting	creating and editing macros
FORTRAN 90	object-oriented	scientific and software engineering
Lisp	functional	artificial intelligence uses - many other languages are derived from it
JavaScript	scripting	writing and enhancing web pages
BASIC	procedural	easy to learn, originally developed to teach non-specialists the art of programming
FORTRAN	procedural	scientific programming language

The above list is not in any way rigorous. There are many instances of particular languages fitting into more than one category. For example, although Visual BASIC is listed as an imperative language it can be used to create event-driven programs, and has many features in common with object-oriented languages.

5.4 Procedural / Imperative languages

A procedural (imperative) programming language tells the computer *how to do something*, written as an ordered sequence of steps that describe exactly what it must do at each step. These instructions, which form the basis of an **algorithm**, are followed in written order by the computer.

Three basic constructs are used to define the order of the steps:

1. *sequence* (the logical ordering of steps);
2. *selection* (a step or sequence of steps are performed if a condition or set of conditions is true);
3. *iteration* (a step or sequence of steps are carried out repeatedly).

Both iteration and selection are *control* constructs because they can alter the *flow of control* of program execution. You will see more of this in later topics.

Examples of procedural languages are Algol, Fortran, Pascal, BASIC, C and COBOL.

Example of a program to read data into an array and output results

Problem: In Visual BASIC we might tell the computer how to input data into an array and then print out the results:

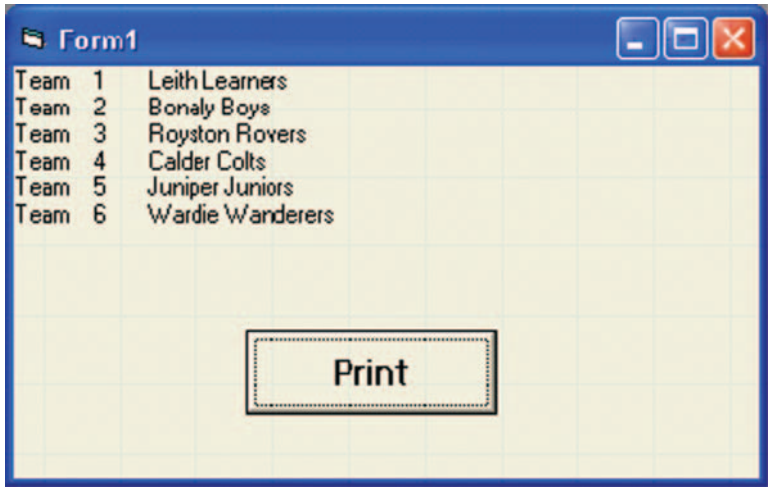
Solution:

```
'Program to fill an array with team names and print them out

Private Sub FillArray_Click()
Dim Team(5) As String
Dim Count As Integer

'Assign array values

Team(0) = "Leith Learners"
Team(1) = "Bonaly Boys"
Team(2) = "Royston Rovers"
Team(3) = "Calder Colts"
Team(4) = "Juniper Juniors"
Team(5) = "Wardie Wanderers"

For Count = 0 To 5
  Print "Teams"; Tab(8); Count+1; Tab(14); Team(Count)
Next Count
Print

End Sub
```

Notice that:

1. the program is expressing at each step precisely how each statement is executed;
2. the program has beginning and end points, which is an indicative feature of procedural language programs.

The program output is:



Structures in procedural languages

On the Web is an interactivity. You should now complete this task.

5.4.1 Review Questions

Q4: One of the main reasons for the many programming languages existing is:

a) older languages can no longer function in the present day computer systems
b) languages have to be adapted so new versions are released
c) there are many different types of operating system
d) people devise new languages because they do not like existing ones

Q5: Which one of the following lists only contains general purpose programming languages?

a) Pascal, C, BASIC,COBOL
b) Algol, Fortran, Prolog, Comal
c) BASIC, Algol, Pascal, Comal
d) Fortran, BASIC, Comal, COBOL

Q6: In computer programming there are three basic constructs. Which one of the following is NOT a programming construct?

a) Sequence
b) Selection
c) Iteration
d) Moderation

Q7: Which one of the following languages could be classified as being event driven?

a) COBOL
b) C
c) Visual BASIC
d) Prolog

Q8: Which of these is **not** true of procedural languages?

a) They employ procedures and functions
b) Programs written in the language are linear in structure
c) Programming instructions are explicit
d) They are low level languages

Sentence completion - languages

On the Web is an interactivity. You should now complete this task.

5.5 Declarative languages

Declarative languages model problem solutions very differently from procedural languages. Programmers specify what the problem is rather than how to solve it. In PROLOG, for example, a program represents knowledge as **facts** and **rules**. Collectively facts and rules are called **clauses**. A proposition is the smallest unit of knowledge that can be judged true or false, such as "*a collie is a sheepdog*", or "*a beagle is a hound*" or "*George passed a ball to Steven*".

In PROLOG these facts would be written as:

```
sheepdog(collie). fact that a collie is a sheepdog
hound(beagle). fact that a beagle is a hound

passed_a_ball(george,steven).
```

A rule contains a condition:

```
Cats purr if they are stroked by humans.
```

This would be expressed in PROLOG as:

`purr(X):- stroked_by(X,humans)` where `X` = cats

PROLOG facts and rules are stored as a database (or knowledge base) which can then be queried to provide solutions.

Example - Using facts, rules and queries

Problem: Suppose we want to find out whether a person drives a fast car. We start by building a set of facts and rules for our knowledge base.

Solution:

```
person(judy). - this is the fact that Judy is a person

person(james). drives_car(james,ford_escort).
```

```
drives_car(judy,porsche). - this is the fact that Judy drives a Porsche

drives_fast_car(X):- drives_car(X,Y) and Y = "porsche"
```

In this example we could ask the program to tell us whether Judy drives a fast car by typing the query:

`?drives_fast_car(judy).` The result would be YES since the goal is satisfied.

If we asked:

`?drives_fast_car(james)`

then the result would be NO as `drives_car(james,Y)` would evaluate `Y="ford escort"`.

This would then cause the rule `drives_fast_car(james)` to fail as `Y` does not equal "porsche" and the goal is not satisfied.

You can see from the code that there is no description of the **type** of data or its internal representation. There are simply statements of facts and a rule.

Contrast this with a procedural language where the programmer would need to set up a structure to hold the knowledge and predefine its type (string, number etc). Then they would need to describe the steps taken to search the structure in order to answer the query. A declarative/logical language is simplistically described as telling the computer *what to do* and *not how to* do it.

Goals and clauses

On the Web is an interactivity. You should now complete this task.

5.6 Event-driven programs

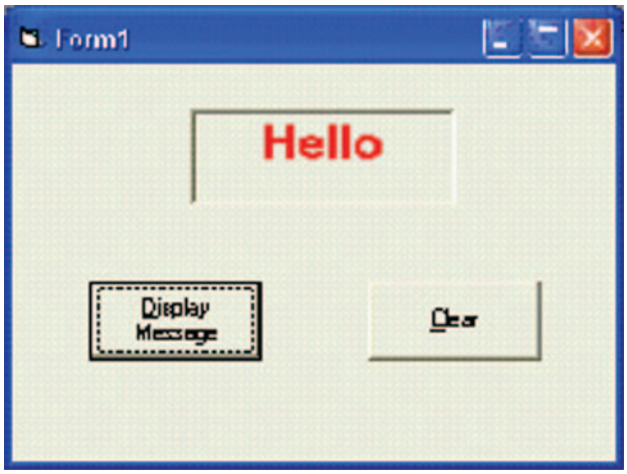
Event driven programming languages have evolved to handle events.
(Strictly speaking, we should talk about event-driven programs, not languages. Some languages, such as Visual BASIC, are ideal for creating event-driven programs, but it is the programs that are event-driven, not the language!)

Events can be initiated at two levels:

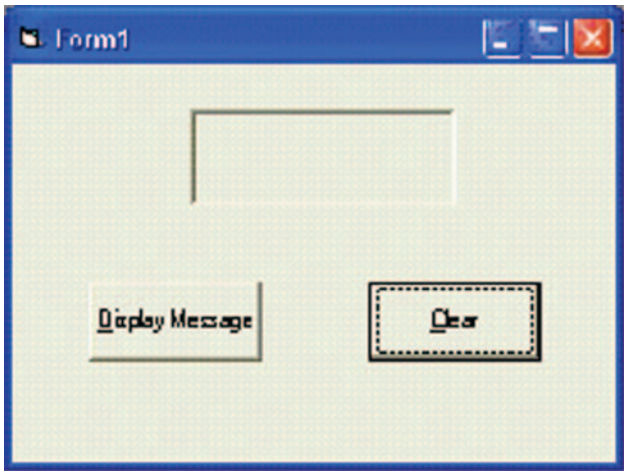
1. at *system/hardware* level: events can include timers, interrupts, loading of files etc;
2. at *language/system* level: events include mouse clicks, keyboard presses and cursor movements.

In Visual BASIC each object has a set of events associated with it. An event is an action that Visual BASIC can detect and respond to. A user clicking on a command button, is an example of an event. The programmer codes the actions that he/she wants to happen when an event occurs.

The following examples show the action of clicking on two buttons in a Visual BASIC program:



Button1 'Display Message' produces a message in the label window



Pressing the 'clear' button removes the message.

A typical event driven program has the effective structure:

```
do (forever)
  if event then
    if event caused by X then
      handle X
    else if event caused by Y then
      handle Y
```

After each event is handled, nothing happens until the next event occurs.

Note that event-driven programs do not have a predefined pathway in the execution of the code, as opposed to imperative programming style i.e. they have no fixed beginning or end.

Graphical user interface programs are typically programmed in an event-driven style using languages such as Visual BASIC and Visual C.

Other languages, such as Java, (an object-oriented language) can also be used to create event-driven Windows-style programs.

Event-driven Programming

On the Web is an interactivity. You should now complete this task.

5.7 Scripting languages

A **Scripting language** is a style of 'programming' that produces ASCII text-based scripts which are usually designed to add functions to, or automate some aspect of, either an application program or an operating system. Scripting languages support high level language control features such as selection and iteration.

Examples of present day scripting languages are *VBScript, JavaScript, Perl, Python and TCL (Tool Command Language)*

Applications that provide scripting capability allows the user to extend the functionality of the application by programming a sequence of actions. For example, in *Filemaker Pro* (a database package) it is possible to write scripts that open and close files, copy data from records or enter a certain database mode such as *browse* or *find*.

An example script in *Filemaker Pro* is shown in Code 1

```
Enter  Browse Mode[]
Go to Layout ["Layout 1"]
Enter Find Mode []
Set Field ["Computing Course", ""S3C8""]
Perform Find[]
Sort [Restore, No Dialog]
Go to Layout["S3 Report Final"]
```

Code 1

Another example is VBScript which is a cut-down version of Visual BASIC, used to enhance the features of web pages in Internet Explorer. Below in Code 2 is an example of a section of VBScript embedded in HTML code.

```
<HTML>
<HEAD>
<TITLE> This is VBScript in action!</TITLE>
<SCRIPT LANGUAGE="VBScript">
MsgBox "Welcome to VBScript"
<P> Click on the button below </P>
<INPUT TYPE ="Button" NAME="cmdClick" VALUE="Click">
</SCRIPT>
```

Code 2

5.7.1 Benefits of scripting languages

One of the main benefits of scripted languages is that they require no compilation. The language is interpreted at run-time so the instructions are executed immediately.

Scripting languages also have a simple syntax which, for the user:

- makes them easy to learn and use
- assumes minimum programming knowledge or experience
- allows complex tasks to be performed in relatively few steps
- allows simple creation and editing in a variety of text editors
- allows the addition of dynamic and interactive activities to web pages

5.7.2 Examples of Scripting Languages

Specialised scripting languages include:

Perl (Practical Extraction and Report Language). This is a popular string processing language for writing small scripts for system administrators and web site maintainers. Much web development is now done using Perl.

Hypertalk. It is the underlying scripting language of HyperCard, while Lingo is the scripting language of *Adobe Director*, an authoring system for develop high-performance multimedia content and applications for CDs, DVDs and the Internet.

AppleScript, a scripting language for the Macintosh allows the user to send commands to the operating system to, for example open applications, carry out complex data operations. An example script is shown in Code 3

```
--this simple Applescript will search
--recursively through folders in a shared volume
--removing certain unneeded files from every user's
--directory

on iterate(thisFolder)
  tell application "Finder
    if the (count of folders in thisFolder) > 0 then
      repeat with innerFolder in every folder of thisFolder as list
        if file "Mail Drop Preferences" of innerFolder exists then
         delete file "Mail Drop Preferences" of innerFolder
        end if
        if file "Email Startup" of innerFolder exists then
          delete file "Email Startup" of innerFolder
        end if
        if file "EmailSavePreferences" of innerFolder exists then
          delete file "EmailSavePreferences" of innerFolder
        end if
      end repeat
    end if
```

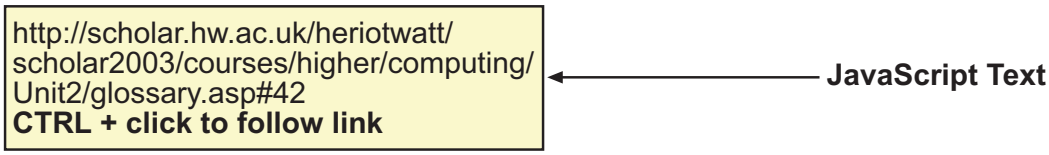
```
  end tell
end iterate
```

Code 3

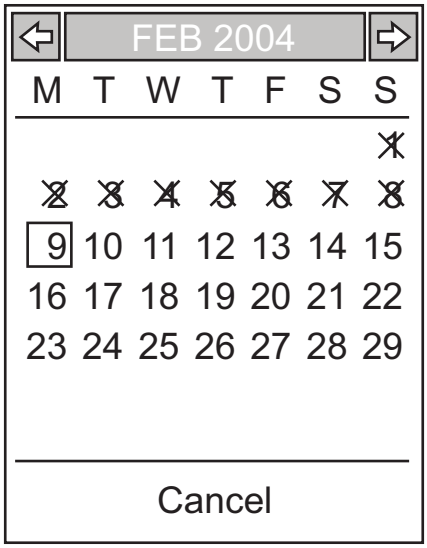
JavaScript is a well-known scripting language that allows more functionality and enhancement to be added to web pages. JavaScript code can run on any platform that has a JavaScript interpreter, which includes most browsers.

Typical uses of JavaScript include:

a) **Image or text rollovers.** If the user rolls the mouse over a graphic or hypertext then a text or graphic box will appear:

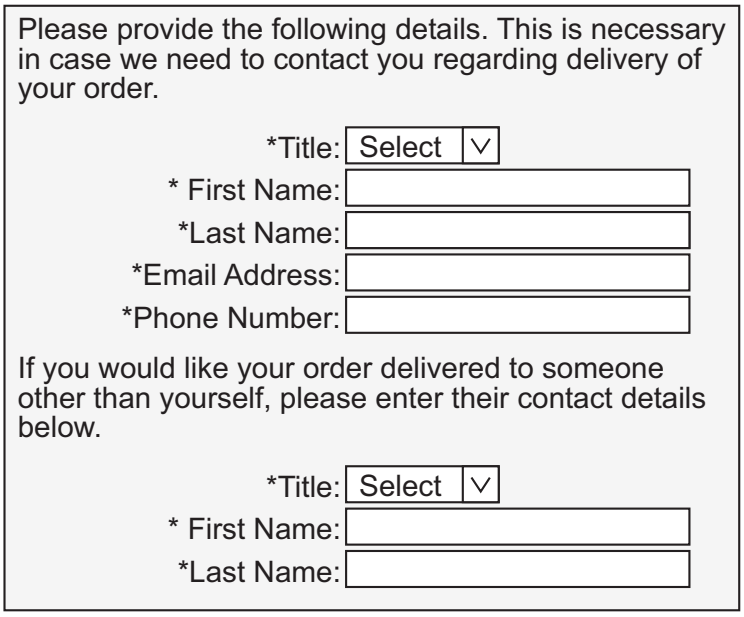


b) **Creating a pop-up window** to display information in a separate window from the Web page that triggered it. This is useful if the user requires to perform a simple calculation or consult a calendar for inputting dates. This is achieved by embedding ActiveX controls or Java applets into the script.



Pop-up Calander

c) **Validating the content of fields.** When filling in forms, each field, especially required fields denoted by an asterisk, are validated for correct input. If the field is left blank or incorrect information entered then a user message will be generated and you may not continue.



5.7.3 The need for scripting languages

Nowadays scripting languages are becoming more popular due to the emergence of web-based applications. The market for producing dynamic web content is now expanding extremely rapidly so new scripting languages have been developed to allow users with little or no programming expertise to develop interactive web pages with minimum effort.

Also the increases in computer performance over the past few years has promoted a comparable increase in the power and sophistication of scripting languages that, unlike conventional programming languages, can even have certain security features built-in. Downloading web-based content from a remote site to a user's local machine can include animations, graphics, MP3 audio files, video clips and so on and this is authenticated by the scripting language.

However be warned! Executable code can inadvertently be downloaded from a remote server to a web browser's machine, installed and run using the local browser's interpreter. This is easily done by visiting dubious web sites or downloading programs without valid authenticity. The user is probably unaware of anything devious occurring. This is a weakness in the formal rules defining scripting languages like JavaScript and VBScript.

5.7.4 Creating a Macro

A **macro** is a way to automate a task that you perform repeatedly or on a regular basis. It is a series of commands and actions that can be stored and run whenever you need to perform the task. Instructions can be simple, such as entering text and formatting it, or complex, like automating tasks that would take several minutes to do manually Macro contents consist of ASCII text and can be created and edited in any simple text editor.

Many programs (like Microsoft Word and Microsoft Excel) can create macros easily. All you have to do is "record" a set of actions as you perform them. For example, you could record opening a new document using a specific template, inserting a header and inserting a name and address and greeting. Each time you "replayed" the macro,

it would perform those tasks. The Macro is stored as a script using the application's scripting language, VBScript.

For programs that don't include a macro facility there are numerous shareware macro programs that can be downloaded and used in any application.

5.7.5 Running A Macro

A macro can be initiated by:

- pressing selected key combination (hot keys)
- clicking an icon on the toolbar that has been created for the macro
- running the macro from the Tools menu of the application.

Example tasks could include:

- inserting your name and address on documents
- formatting text with specified font and size
- accessing websites from a list of 'favourites'
- inserting special symbols or graphics into documents
- automate playing of audio CDs while you work on the computer
- formatting of spreadsheet cells
- creating headers and footers.

Code 4 is an example VBScript listing for a macro to create a table in Microsoft Word:

```
'Macro to create a table in Word
'
Sub Macro1()
'
' Macro1 Macro
' Macro recorded 05/01/2004 by John Smith
'
ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=7,
NumColumns:= _
  4, DefaultTableBehavior:=wdWord9TableBehavior,
AutoFitBehavior:= _
  wdAutoFitWindow
With Selection.Tables(1)
  If .Style <> "Table Grid" Then
    .Style = "Table Grid"
  End If
  .ApplyStyleHeadingRows = True
  .ApplyStyleLastRow = True
```

```
    .ApplyStyleFirstColumn = True
    .ApplyStyleLastColumn = True
  End With
End Sub
```

Code 4

5.7.6 Review Questions

Q9: An event-driven language differs from other languages in that (choose one):

a) Programs have no pre-defined pathway
b) Programs are initiated entirely from mouse clicks
c) Program events cannot be nested
d) Programs are difficult to write

Q10: Which of these is **not** a benefit of scripting languages?

a) They are easy to learn and use
b) Allow complex tasks to be performed in relatively few steps
c) Web pages can be made more dynamic
d) They describe a problem rather than how to solve it

Q11: Many software applications include a macro facility. Which one of the options best describes a **macro**?

a) Macros are not easy to edit
b) The creation of a macro can be time-consuming
c) Macros automate repetitive tasks
d) A macro listing consists of complex code

Q12: One of the tasks that a macro could NOT perform would be:

a) Inserting special symbols or graphics into documents
b) Formatting a floppy disc
c) Creating headers and footers
d) Changing audio CDs

5.8 Other Language Types

Other important language types include object-oriented languages and functional languages. You do not need to know about these for Higher Computing.

5.9 Translation methods

At the end of the implementation stage, if all is going well, a structured program listing will be produced, complete with internal documentation. This will be thoroughly checked against the design and against the original specification.

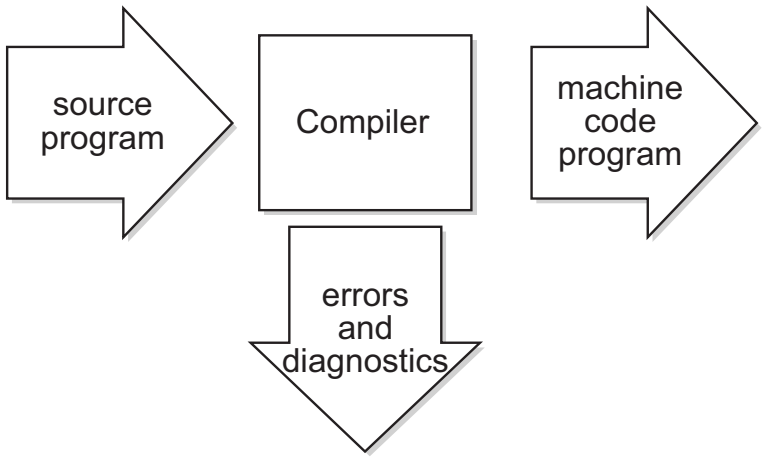
The high-level code written at this stage is called **source code** which must be translated

into machine code, called **object code** that the computer understands.

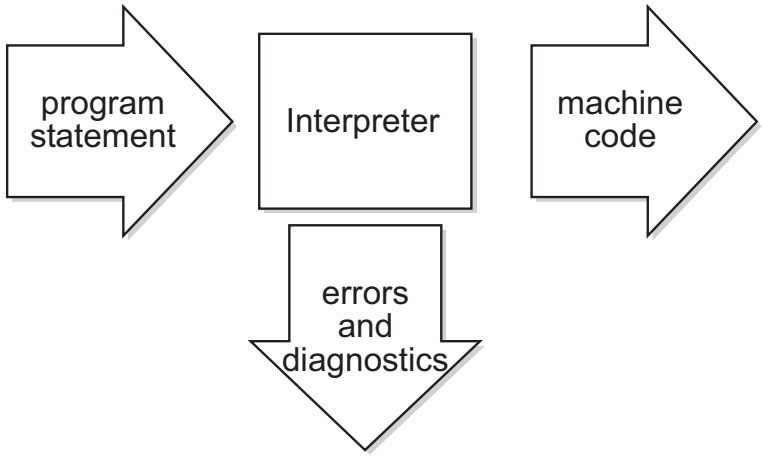
There are two methods of translating source code into object code; a compiler and an interpreter.

5.9.1 Compilers and Interpreters

A compiler, which is a complex program in itself, translates source code into object code that is then loaded into main memory and executed.



Another form of translation that converts source code into object code is an **interpreter**.



Unlike a compiler, an interpreter checks syntax and generates object code one source line at a time. Think of this as very similar to a group of translators at a United Nations' Conference, who each have to convert sentences spoken by delegates into the native language of their representative.

When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter executes each line of code at a time the programmer is able to see the results of their programs immediately which can also help with debugging.

5.9.2 Comparing compilers and interpreters

Efficiency

The main difference between an interpreter and a compiler is that compilation requires analysis and the generation of machine code only **once**, whereas an interpreter may need to analyse and interpret the same program statements **each time** it meets them e.g. instructions appearing within a loop.

For example, consider the following Visual BASIC code:

```
For iCountvar = To 20
iSum = iSum + iCountvar
Pic.Display iSum
Next iCountvar
```

Using a compiler, the source code would be analysed and compiled into machine code once only.

Using an interpreter, the source code would be converted into machine code 200 times (once each time round the loop).

Errors

This has implications for error reporting. For instance, when the interpreter encounters an error it reports this to the user immediately and halts further execution of the program. Such instant feedback, pinpointing the exact location of the error, helps the programmer to find and remove errors.

Compilers, on the other hand, analyse the entire program, taking note of where errors have occurred, and places these in an error/diagnostic file. If errors have occurred then the program cannot run. Programmers must then use the error messages to identify and remove the errors in the source code.

Some compilers assist by adding line numbers to the source listing to help pinpoint errors and all compilers will describe the nature of the error e.g. missing semi-colon, expected keyword, etc. - although interpreting some compiler diagnostics is a skill in itself.

Ease of use

Interpreters however are more suitable for beginners to programming since errors are immediately displayed, corrected by the user, until the program is able to be executed.

On the whole compilers tend to be more difficult to use.

Portability

A compiler produces a complete machine code program which can be save, copied, distributed and run on any computer which has the appropriate processor type.

An interpreter does not do this. The machine code has to be generated each time the program is run. This means that the interpreter must always be present, and program execution is slower.

5.9.3 Review Questions

Q13: One of the main differences between a compiler and interpreter is:

a) An interpreter is faster then a compiler
b) A compiler is better for beginners
c) A compiler creates an independent machine code program
d) An interpreter is much harder to use

Q14: One disadvantage of using an interpreter is:

a) Looping structures have to be interpreted each time they are entered
b) It stops execution when an error is encountered
c) It helps the user to debug programs
d) An interpreter is ideal for beginners

Q15: High level languages have to be translated because:

a) Computers can only understand machine code
b) Source code is faster to run than object code
c) Programs run faster when converted to binary
d) All of the above

5.10 Summary

The following summary points are related to the learning objectives in the topic introduction:

- there are many types of programming languages in use today, including procedural, declarative, event-driven and scripting;
- they are difficult to organise into discrete categories because of overlapping properties;
- high level languages have to be translated to machine code by compiler or interpreter;
- scripting languages can be used to create and edit macros.

5.11 End of topic test

An online assessment is provided to help you review this topic.

Topic 6

High Level Language Constructs 1

Contents

Prerequisite knowledge

Before studying this topic you should be able to describe and use the following constructs in pseudocode and a suitable high level language:

- *input and output;*
- *numeric and string variables;*
- *assignment statements;*
- *arithmetical operations (+, -, *, /, ^);*
- *logical operators (AND, OR, NOT)*
- *conditional loops using fixed and complex conditions;*
- *IF statement;*
- *nested loops.*

Learning Objectives

After completing this topic, you should be able to:

- *describe the need for programming variables;*
- *describe and use real, integer, string and boolean variables;*
- *describe string operations such as concatenation and substrings;*
- *distinguish between local and global variables;*
- *understand what is meant by the scope of variables;*
- *understand the nature and use of sub procedures;*
- *understand the use of selection in programming an in particular the CASE construct.*

Revision

Q1: One of the following program statements produces the value 13 for the variable Answer. Which one?

a) Answer = 5 + 8 * 3 - 2
b) Answer = (5 + 8) * 3 - 2
c) Answer = 5 + (8 * 3) - 2
d) Answer = 5 + 8 * (3 - 2)

Q2: Consider the following segment of programming code:

```
For x = 1 to 3 For y = 1 to 3 Sum = x + y Next y Next x Print Sum
```

When run what would be the final value of the variable Sum?

a) 2
b) 4
c) 6
d) 8

Q3: Which one of the following statements represents a valid string assignment?

a) Name = "Bert"
b) Number = "12345"
c) Paper = "Scots" + "man"
d) All of the above

6.1 Introduction

In this topic you will be introduced to the Visual BASIC programming environment. Variables and data types are discussed, together with fundamental programming structures. Formatted input and output constructs are described and exemplified using Visual BASIC code. More advanced techniques involve conditional statements such as nested IF statements and the CASE construct. Each Visual BASIC construct is exemplified using pseudocode and high level equivalent. .

6.2 The Programming Environment

In this topic, all language features are exemplified using Visual BASIC 6.

Users of Visual BASIC.NET should find few incompatible problems. Important differences will be flagged, where appropriate.

Visual BASIC 6

Visual BASIC is an event driven programming language that works within the graphical environment of Windows. Visual BASIC code is associated with objects and each object has a set of events associated with it.

Events are actions which Visual BASIC detects and respond to. For example:

- a user clicking on a command button on a form will generate a click event for that button;
- pressing a key on the keyboard could initiate a load event for a programming module.

When an event occurs Visual BASIC will run any code that you have entered for that event. The code for each event must be entered as a separate sub program.

Visual BASIC Environment

When Visual BASIC is launched using the Standard EXE option a number of separate windows appear as seen in Figure 6.1 (Note: the layout of the windows on the screen may be different from the example shown - they can be re-arranged to suit your own preference)

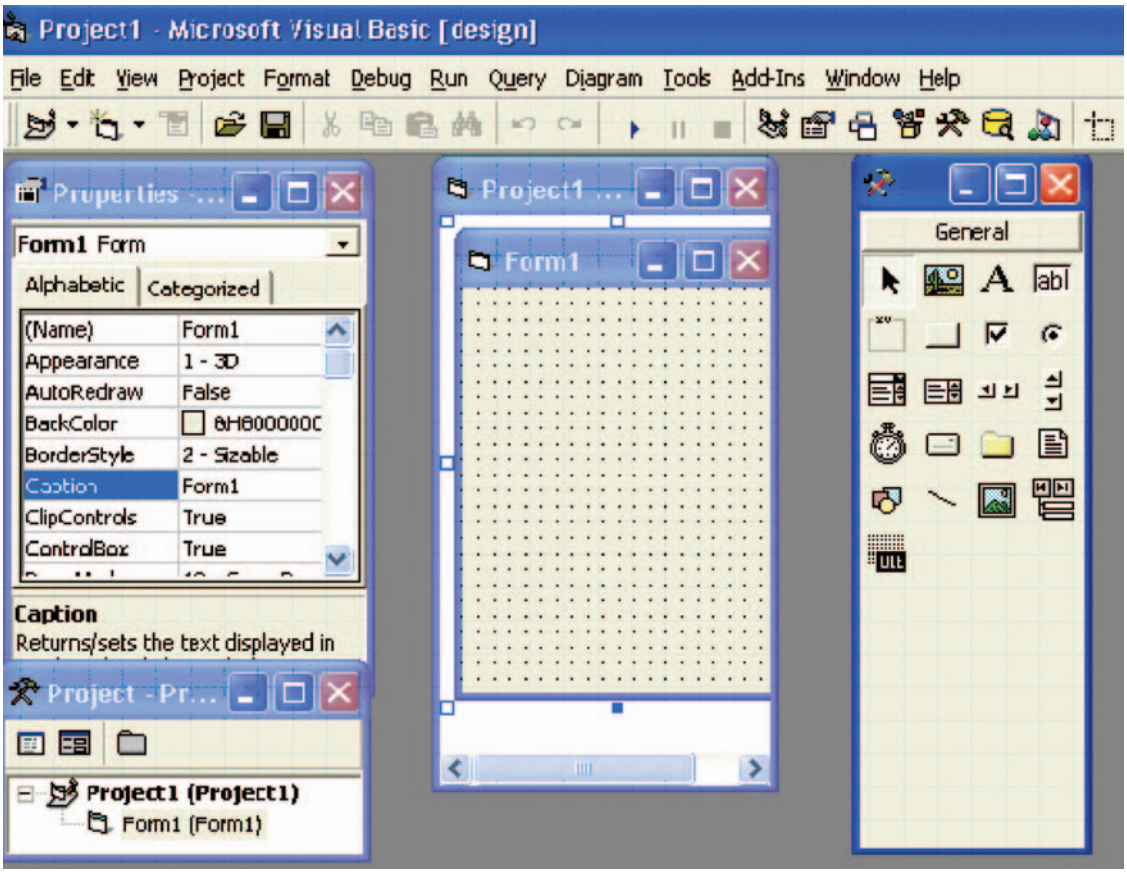


Figure 6.1:

The default screen that you can see is in design mode and consists of four main windows:

1. a blank Form window which is the interface with the application (program) you create. The visual design of the program is created on this form, which has gridlines to help build objects like text boxes, labels and control buttons etc. What

is placed on a form will be displayed in a window when the program is executed.

2. a Project window that displays the files that are created during the construction of the program. These files could be forms, modules (blocks of code not attached to a form), graphics, or control structures such as Active X, required for the successful running of your program. Note that only one project can be open at any given time.

3. a Properties window that displays the properties of the objects created in the program. The form window itself has properties associated with it.

4. a Toolbox window that consists of all the controls necessary for developing a program. Boxes, labels, buttons, and other objects can be drawn on the form as part of the visual interface and also to allow input and output of data.

At this stage it is important to realise that when you create a program, each form, module, graphic, and ActiveX control is saved as an individual file.

Table 6.1 shows the common files types in a Visual BASIC Project:

Table 6.1:

File Type	**Description**
FRM	Form
BAS	Module
OCX	ActiveX control
CLS	Class module
VBP	Visual BASIC project

When starting to write Visual BASIC programs the two most common file types are *Form* and *Project*.

6.3 Building applications

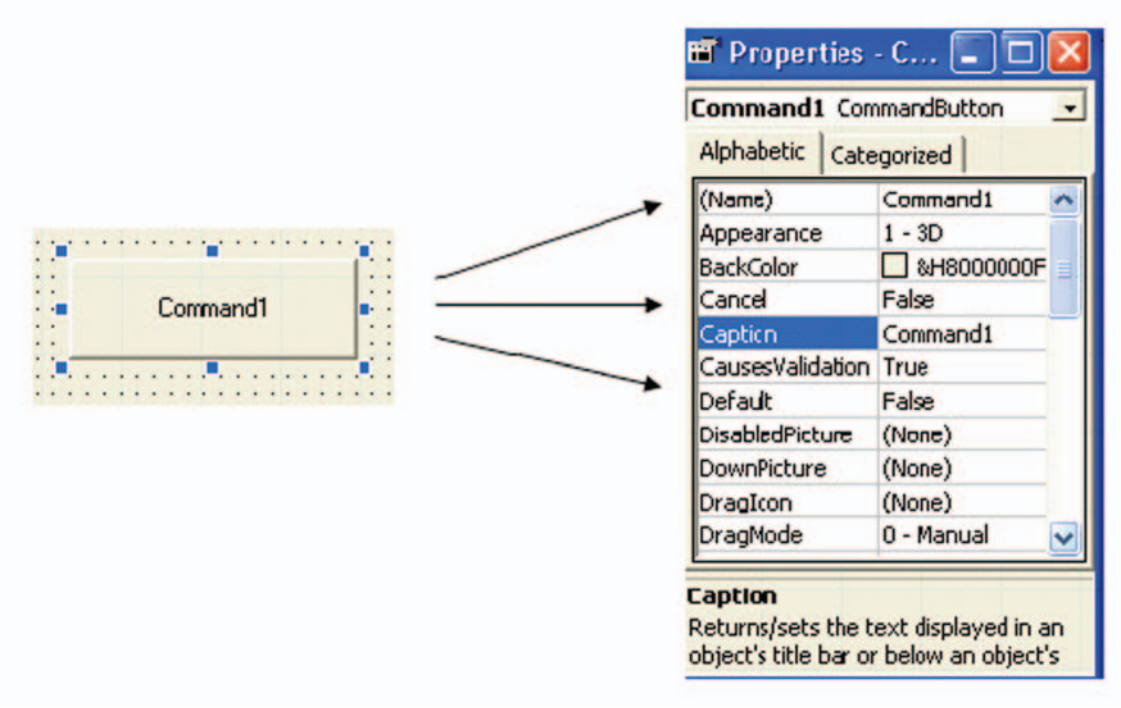
Constructing a program in Visual BASIC involves two processes:

1. creating the visual design of the program,

2. designing and implementing programming code.

6.3.1 Creating a form and objects

In the first process the properties of the form are established followed by the creation of controls inside the form. The properties of each control are then established.



Pressing function key F4 on an active button will open the Properties box as seen above.

Figure 6.2 shows five objects: 3 control buttons, a message box, and a form.

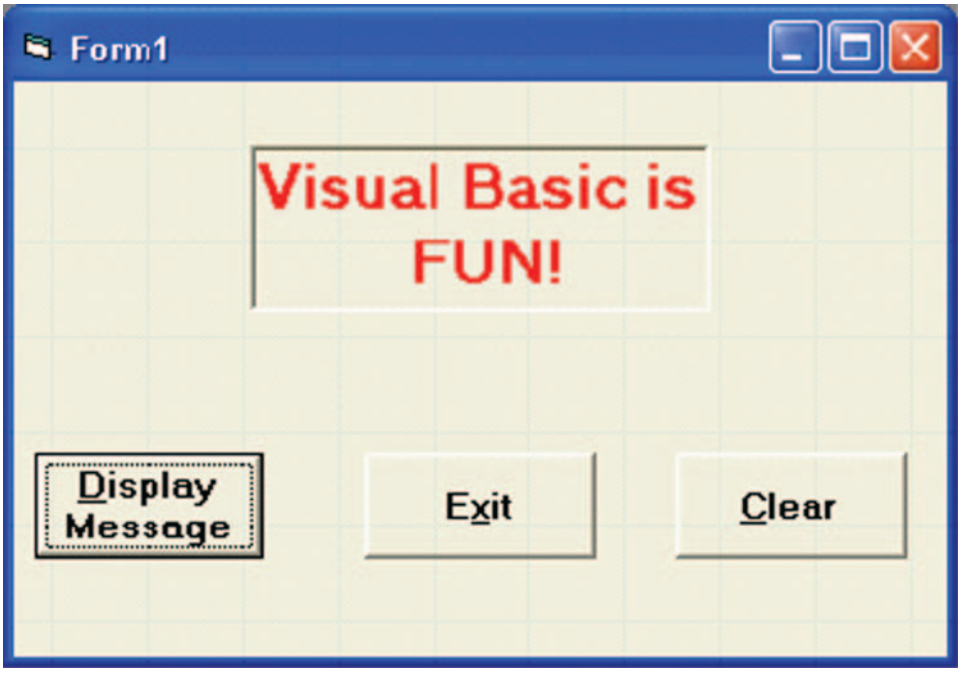


Figure 6.2:

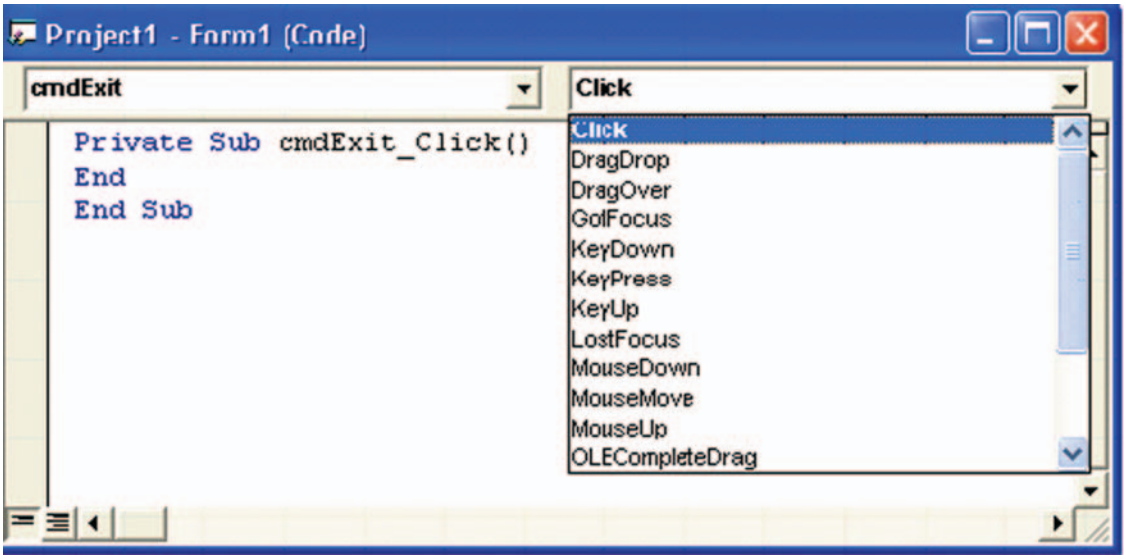
6.3.2 Attaching code to objects

In the second process programming code is written for each event and attached to the appropriate object in the form.

The code for each event is written in the form of a sub procedure.

A sub procedure or sub program is a block of code that is implemented when an event, such as a mouse click occurs.

Double-clicking on any of the objects in a form or even the form itself will open up the program editor and reveal the code for each event. For example clicking on the *Exit* button will display the code editor as follows:



All Visual BASIC code is written between the lines that Visual BASIC enters for you:

```
Private Sub cmdExit_Click() End Sub
```

On the right of the screen you can see a list of other events that are part of the Visual BASIC environment.

6.3.3 Input and Output Controls

Visual BASIC programs do not output to the computer screen as such. Instead they use a *form* which is, essentially a Visual BASIC window that is the interface between the user and the application and allows information to be input by the user or displayed to the user. There may be more than one *form* in a Visual BASIC project.

Information may be input to a program via the keyboard or read directly from data files. The program may also contain data that is assigned directly to variables or constants during execution time.

The output of information may be achieved through a variety of Visual BASIC control structures that are placed on the form at the design stage. Output can also be directed to a printer or to an existing file.

Since file I/O is beyond the scope of the Higher Computing course, only the graphical methods will be discussed at this stage.

The stages in writing a program in Visual BASIC involve the following aspects:

1. **Design stage**: the form is used to compose the graphical elements of the program. Command buttons and other objects can be placed anywhere on the form and these will dictate how the application is to run.
2. **Setting properties**: the form itself and the objects it contains can have various properties set, producing the visual effects required of the program. Although there are numerous properties for each object the main ones would include:
   - Name of the object;
   - Colour;
   - Caption attached to object ;
   - Height/width;
   - Font used;
   - Position of object within the form;
   - Border style.

   Properties are changed by highlighting the object and pressing function key F4. This opens up the Properties window from which changes can be made.
3. **Coding stage**: code is designed and written for each object and when the main program is run each event will become part of the overall effect. Example events would include:
   - Clicking a command button;
   - Loading a form;
   - Clearing the contents of a text box;
   - Ending a program.

When Visual BASIC is started, *Form1* shows by default.

You will see exemplar programs that make use of the following Visual BASIC constructs later in this topic and others.

6.3.4 Input Methods

In Visual BASIC there are many ways of inputting data. We will only consider two of these:

- using the InputBox function,
- using Text Boxes on the form.

6.3.4.1 InputBox function

A program may ask for user input during its execution. This can be accomplished using the InputBox function, which by default allows text entry.

The full syntax for the InputBox function is:

```
InputBox "Prompt", "Title", xpos, ypos
```

`Prompt`: is the displayed message
`Title`: is the optional text that will appear in the title bar
`xpos,ypos`: coordinates for positioning the InputBox on the screen.

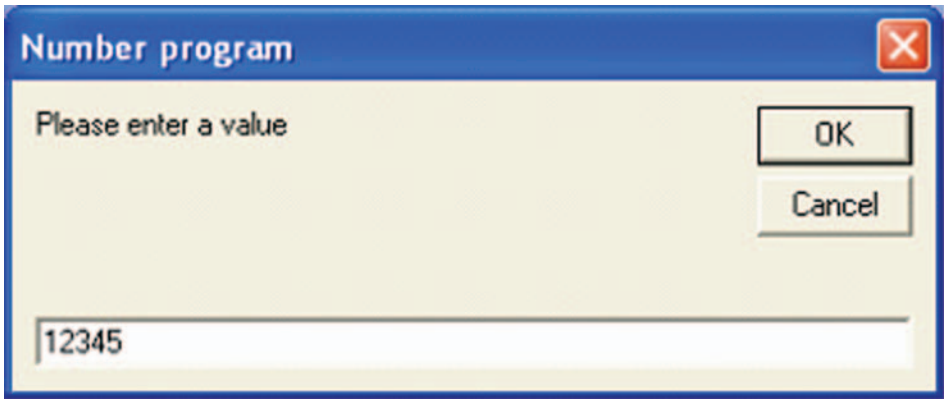
For example a program may require the user to enter a name. The format is:

```
Name = InputBox("Please enter a name",)
```

The user then complies with the request to enter a name.

Should a numerical value be required by the program then the input format is identical but a numeric variable is used.

```
Number1 = InputBox("Please enter a value","Number                    program", ,50,50)
```

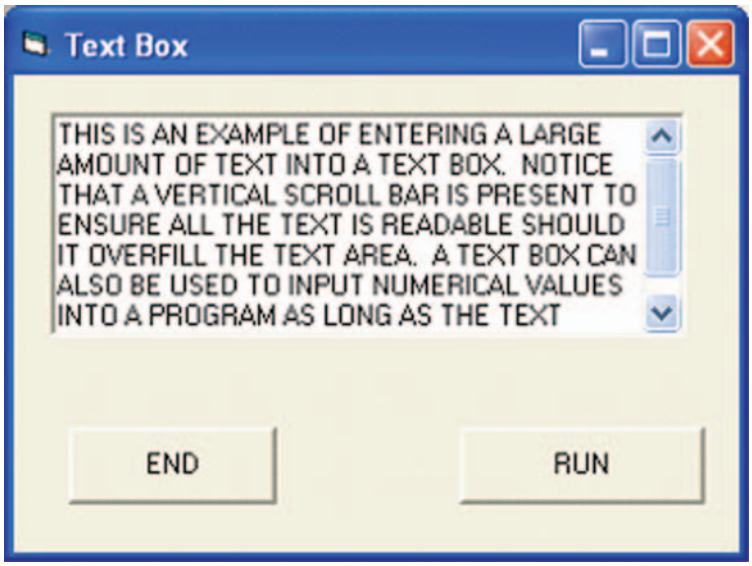


In this case the value `12345` is input to the program, a caption appears in the title bar and the InputBox appears at coordinates 50, 50

In most cases only the prompt is used.

6.3.4.2 Use of the Text Box for Input

A text box can be used instead of an input box. The main advantage of a text box is that a user can type in much more information during the execution of a program. Since text boxes can hold large amounts of text it is best that they are created using a vertical scroll bar.



In the properties window, the options `Vertical scroll bar` and `Multiline = True` are set.

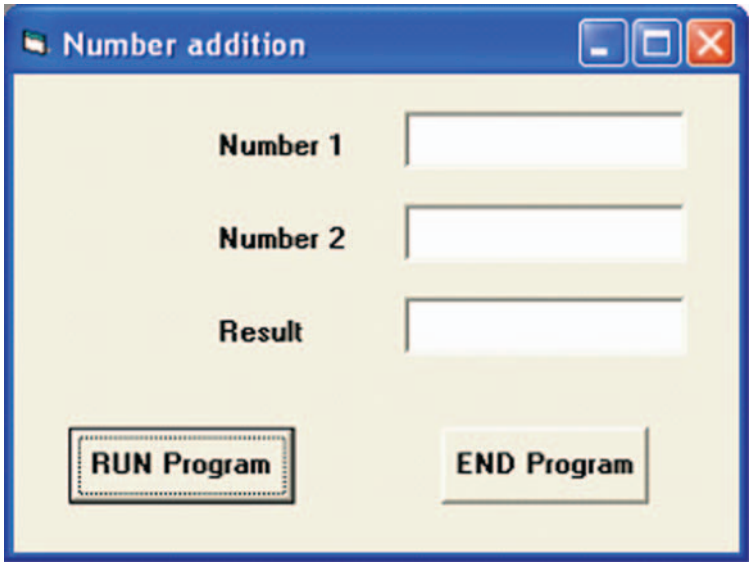
6.3.4.3 Exercises on Input Methods

Exercise 1

Open a new Visual BASIC project and experiment with adding buttons and text boxes to the form from the toolbox window. Use the properties window to try-out various settings and captions for the buttons and settings for the text box and form.

Exercise 2

Open a new Visual BASIC project and produce the following form that shows text boxes, labels and buttons. Use the properties window to create captions for the objects.



6.3.5 Output Methods

In Visual BASIC there are many ways of outputting data. We will only consider two of these:

- using the MsgBox function,
- using the Print command.

6.3.5.1 The MsgBox

A message box allows the program to display any message to the user in a pop-up window on screen. The `MsgBox` function is useful in situations where the user requires, for example, to confirm an action. In its simplest form the format of the function is:

```
MsgBox "The value entered was within the required range"
```

The output screen would look like:

The full format of the `MsgBox` statement is:

```
MsgBox "Prompt", Buttons,                    "Title"
```

`Prompt`: is the message to be displayed in the MsgBox
`Buttons`: is a numeric expression that specifies which buttons to display, with or without icons.
`Title`: The string message displayed in the title bar.

Exercise 3

Experiment with MsgBox button values and create your own messages.

6.3.5.2 The Print command

The print statement is a useful output command since it can be used on its own or in combination with other objects and functions.

Used on its own the print statement with no explicit destination will produce output to the current window which, in most cases, is the Visual BASIC form.

For example, if X = 5 and Y = 7 the following statement will output a text string and value to the form:

```
Print "The sum of the two variables X and Y is "; X + Y
```

If the form is populated with many objects it may be difficult to see the above output.

It is much better, therefore if the output is directed to a specified window such as a MessageBox, TextBox or PictureBox. Although a PictureBox is meant for graphics it is also ideal for the output of lists of text. Using a PictureBox the previous print statement would become:

```
PictureBox.Print "The sum of the two variables X and Y is "; X + Y
```

Usually the term PictureBox is changed to PicBox, PicOutput, PicDisplay, Picture1 or whatever name is suitable for the output.

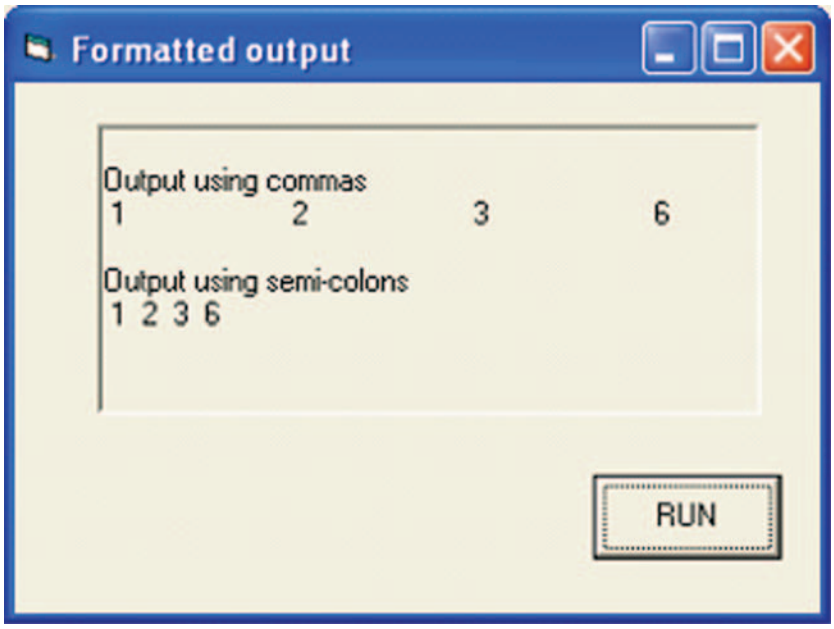
Exercise 4

Create a new project and place a command button and PictureBox on the form. Rename the PictureBox as PicDisplay, and the command button as cmdAdd.

Double-click on the command button and enter the following code. The first and last lines are present by default so they can be ignored when entering the text.

```
Private Sub cmdAdd_Click()
  Dim number1 As Integer, number2 As Integer, number3 As Integer
  Dim Sum As Integer
  number1 = InputBox("Please enter a value")
  number2 = InputBox("Please enter a value")
  number3 = InputBox("Please enter a value")
  Sum = number1 + number2 + number3
  PicDisplay.Print
  PicDisplay.Print "Output using commas"
  PicDisplay.Print number1, number2, number3, Sum
  PicDisplay.Print "Output using semi-colons"
  PicDisplay.Print number1; number2; number3; Sum
  PicDisplay.Print
End Sub
```

Run the program using your own values of number1, number2 and number3. Your output should look like the following:



Use of TAB() and SPC()

Two valuable print functions are TAB() and SPC() that allow for more formal results.

`TAB(6)` will begin output '6' units from the left margin

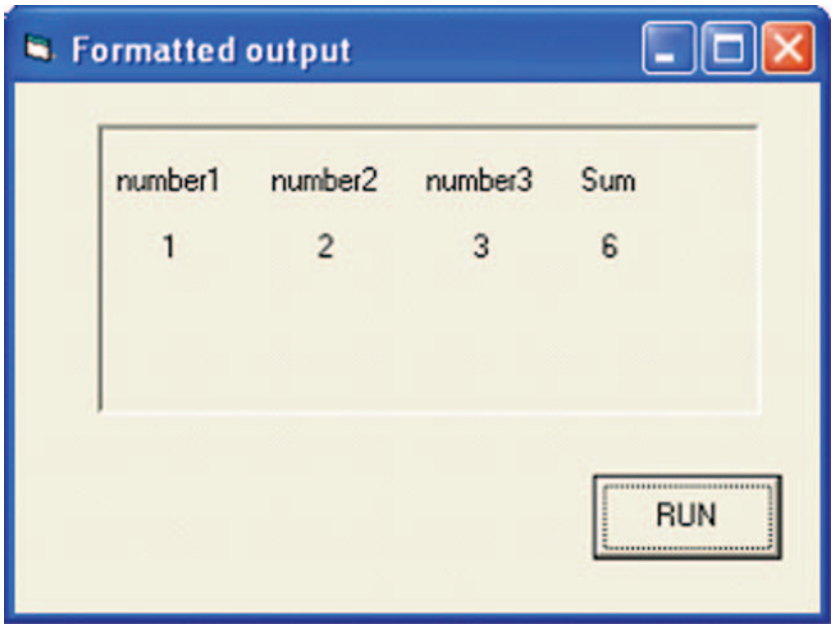
`SPC(6)` will output '6' units from the previous output.

Exercise 5

Modify the previous code as follows and run the program:

```
PicDisplay.Print
PicDisplay.Print  Tab(2); "number1"; Tab(14); "number2"; Tab(26);
  "number3"; Tab(38);
  "Sum"
PicDisplay.Print
PicDisplay.Print Spc(4); number1; Spc(9); number2; Spc(9);  number3;
   Spc(7); Sum
```

This time you end up with the output that should look like the following:



You will learn more about formatting values later.

6.4 Program Structure

A typical Visual BASIC program takes the form shown in Code 5.

```
Option explicit
'General area
'Public or Global variables declared here
    ------------------------------------

Private Sub cmdClick_Click()
constant definitions
  'the constants used in the program

variable definitions
  'the variables used in the program
  'main parts of the program

INPUT PHASE
  'read in data from keyboard, file, , mouse etc

CALCULATION
  'process the data to give us the results we want

OUTPUT
  'send results to a window, printer, file, disk etc.
```

```
End sub
    ----------------------------------

Sub procedures()
  'Procedure and function definitions

End Sub
```

Code 5

The program area contains the following:

- A *general area*: this area is public and variables declared here will be 'seen' by all parts of the program. Global variables should be avoided wherever possible. More on this will be discussed later under the term *scope*.
- A *sub procedure cmd_click( ):* this contains the main program code that will be initiated in the event of a mouse click, for example. Within this code there are further declarations local to the sub procedure. The declarations are:
  - *Constants*: any fixed values which will stay *constant* throughout the program are declared here;
  - *Variables*: variables other than global variables that are going to be used in the program are declared here.
  - *Sub procedures*: any subprocedures required by the program are declared after the main sub procedure. These are described more fully later on.
  - *Functions*: - any functions the program will use are also declared here. These are described more fully later on.

The input may be from a variety of devices, not just the keyboard. A disk is an input device when the computer reads data (including programs) from it. The program may also receive data internally during program execution.

The calculation is the heart of the program. The input data is transformed to the output data. This, in short, is all that a computer does.

The output phase is less straightforward! There is no screen output in Visual BASIC, only windows called forms. Output can also be directed to a file, printer or disc.

You will see example output methods in the program exemplars.

6.4.1 Example program

Suppose, for example that you were going to write a program to calculate the area of a rectangle, where the user will input the dimensions and the program will return the area. The program will take the general form as shown in Code 6.

```
Option Explicit
    ---------------------------------

Private Sub cmdDisplay_Click()

  Dim Length As Integer
  Dim Width As Integer
  Dim Area As Integer

  Length = Lngth.Text 'Input length
  Width = Wdth.Text 'Input width
  Area    = Length * Width 'Calculate area
  Ar.Text = Area 'Output result

End Sub
```

Code 6

This program uses text boxes to input and output the data.

1. In the program the variables *Length, Width* and *Area* are declared using the `Dim` statement (see later)
2. Values of Length and Width are input via text boxes.
3. The area is calculated.
4. Result output to a text box.

Figure 6.3 shows a program run:

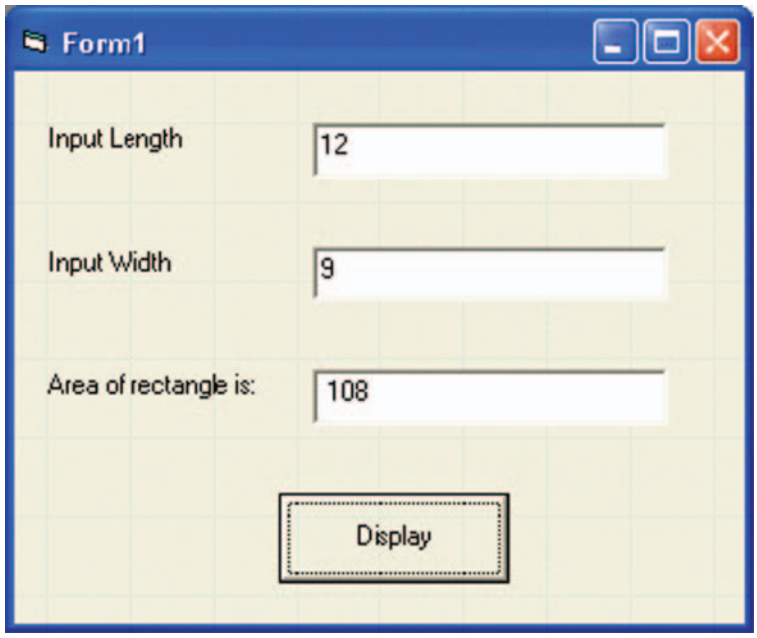


Figure 6.3:

You can see that the form contains 3 labels and 3 text boxes for the input and output of data and a control button.

6.5 Data types

Visual BASIC has a number of in-built data types, the significant ones for this topic being integer, single (real), boolean and string.

These are summarised with others in Table 6.2

Table 6.2:

Data Type	Description
`Integer`	These are whole numbers, positive or negative, without a decimal point. The range depends on the machine but if 16 bits long the range is -32,768 to +32,767.
`Long (integer)`	Extended range of integer from -2,147,483,648 to +2,147,483,647
`Single (real)`	These are floating point numbers, e.g. 3.33333. When 32 bits long their range is $\pm$3.4e $\pm$38, and referred to as *single-precision*.
`Double (real)`	*Double-precision* floating point numbers within the range $\pm$1.7e $\pm$308.
`String`	Fixed length up to 65,400 characters Variable length up to 2 billion characters.
`boolean`	This data type can represent two values: *true* and *false*.
`variant`	Any numeric value up to the range of a double-precision number or any character text.

Note that:

1. Although the range of floating point numbers is huge, they are not all that accurate - often not much more than 8 - 16 digits (depends on the machine and compiler). The rest of the number you may see on the screen is an approximation. For accuracy you need to use integers.
2. The data type 'variant' is the default mode in Visual BASIC and will be discussed later in the topic. It should **not** normally be used.

Identifying data types

On the Web is an interactivity. You should now complete this task.

6.6 Naming variables

An important aspect of any language is the rules for naming the objects such as *constants, variables, subprocedures* and *functions*.

There are *five* rules in Visual BASIC which prescribe *valid* variable names. All Visual BASIC variable names:

1. must begin with a letter
2. must contain no spaces
3. must only consist of letters or digits with no punctuation marks except the underscore character
4. can be no longer than 255 characters
5. cannot use Visual BASIC reserved words.

   Some of these keywords are listed in Table 6.3 and ***must not*** be used as variable names.

Table 6.3:

`dim`	`else`	`true`	`false`	`to`
`sub`	`function`	`while`	`wend`	`private`
`const`	`integer`	`for`	`string`	`elseif`
`case`	`public`	`repeat`	`program`	`and`
`procedure`	`boolean`	`until`	`clear`	`or`
`if`	`string`	`do`	`print`	`not`

A more complete list of keywords can be found in any good Visual BASIC manual.

The variable names shown in the table are valid:

temp1	velocity	unit1
pay_period	fred	day_month_year

The variable names shown in the table are **not** valid. Why?

40th_birthday	@discount	%_rate
const	bad-variable	string

Online there is an interactivity "Naming variables". You should attempt this now.

Sentence completion - variables

On the Web is an interactivity. You should now complete this task.

6.6.1 Review Questions

Q4: Which one of the following variable names would not be allowed in Visual BASIC?

a) Hello
b) Number_1
c) 17.5%Vat
d) SCHOLAR

Q5: The choice of a meaningful variable name in programming is very important because:

a) It makes it easier for programmers to locate and fix errors
b) It makes the programs more efficient
c) Meaningful variable names make the program more reliable
d) All of the above

Q6: A program is written to input a person's age in years and calculate how old they will be in 10 years time. Which of the following data types would be required in the program to hold the person's age?

a) String
b) Real
c) Boolean
d) Integer

Q7: A program is designed to generate prime numbers up to the maximum value possible with precision. The results are stored in a variable called *prime*. In order to perform the calculation the variable prime would need to be declared as:

a) Integer
b) Real (double)
c) Integer (long)
d) Real(single)

6.7 Declaring Variables

Within Visual BASIC variables are used to represent and identify values within a program. The values are held in temporary storage locations in the computers memory.

Each memory location is identified by a unique variable name, and the value of its contents can change during the execution of a program.

Once the program has ended variables will be reset; all numeric values become zero and strings become empty.

For this reason it is important that all variables are declared before a program is run.

To use a variable in Visual BASIC, three quantities must be specified:

1. Name of the variable
2. Type of variable
3. Value of the variable

Variables are declared using the `Dim` statement which allocates temporary storage to them. It is usual to use `Dim` statements before any other code so that total memory can be allocated at run time.

Examples of Dim statements are:

```
Dim   Found As boolean
Dim   Maximum As integer
Dim   Precision As double
Dim   Myname As string
```

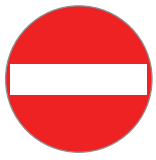
6.7.1 Implicit and Explicit declaration

VB6 offers two levels of variable type declarations:

1. Implicit declaration
2. Explicit declaration

Implicit declaration (to be avoided)

In **implicit declaration**, if the 'dim' statement is used on its own without assigning type to a variable or a variable is assigned a value, then Visual BASIC will assign the data type variant. This is the default setting that Visual BASIC will assign variables if not declared as some other type.



Visual BASIC does not force you to declare data types but it is **much better** if you do.

For example consider the following lines of code as shown:

```
Private Sub Form_Load()
```

```
Dim MyText
MyText = "Sample program"
FirstNumber = 5
SecondNumber = 10
Total = FirstNumber + SecondNumber
End Sub
```

FirstNumber, SecondNumber and *Total* have not been declared but Visual BASIC has assigned them data types 'on the fly' the first time they encountered. Although *MyText* has been declared, it has no type. All variables have therefore been assigned the type *variant*.

But beware! If a *variant* variable is mis-spelled later in the program then a new variable will be created by Visual BASIC. This is a common error and can create program bugs that are difficult to find.

Explicit declaration (to be encouraged)

In **explicit declaration** each variable is declared unambiguously using the `Dim` statement.

It is recommended that this option be used at all times. This helps to prevent errors and allows the computer to work more efficiently. If Visual BASIC knows the data type through declarations then the requisite amount of memory can be assigned thus making memory management more efficient.

Visual BASIC can be forced to make variable declaration the default setting. By clicking on *Tools* then *Options* a window like Figure 6.4 will open. Checking the *Require Variable Declaration* box will ensure Visual BASIC starts up in explicit declaration mode.

Note: Visual BASIC.NET does not support the type variant.

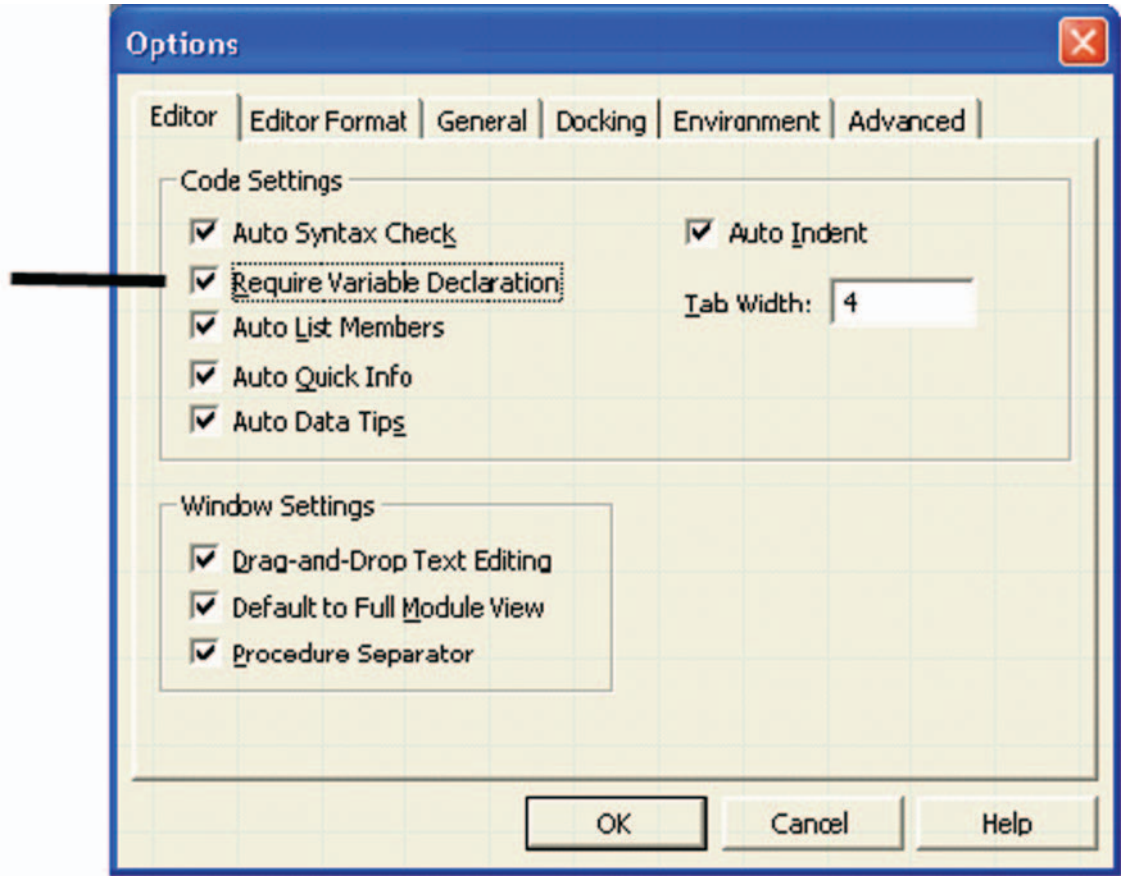


Figure 6.4:

Here is the simple program to add 2 whole numbers, illustrating proper declaration of variables.

```
Option Explicit

Private Sub Form_load()

Dim FirstNumber As integer
Dim SecondNumber As integer
Dim Total As integer
Dim MyText As string

FirstNumber = 5
SecondNumber = 10
Total = FirstNumber + SecondNumber

MyText = "Sample Program"

End Sub
```

Code 7

Q8: What is wrong with this Visual BASIC program?

```
'program subtract

Dim number1 As Integer
Dim result As integer

number1 = InputBox("Please enter the first number")
number2 = InputBox("please enter the second number")
result = number1 - number2
Print("Number 1 take away number 2 = ";sum)
```

Calculating minutes


20 min

Write a program which prompts the user to enter a number of days, hours and minutes. The program will then calculate and display this as a total number of minutes.

6.8 Declaring constants

A constant is a quantity that is allocated by the user, usually at the start of a program, although they can be defined anywhere in the program. A **constant** retains the **same value** throughout program execution and cannot be altered.

Constants are declared using the const statement to create string or numeric values.

Examples of constant declarations are:

```
Const cPi As single = 3.141592

Const cAvogadro   As single = 6.023E23

Const cName As string= "This is my name"
```

Some programmers like to express constants all in capital letters to differentiate them from variables. This is an individual preference. This makes no difference to the program whatsoever; its only purpose is to bring to your attention the fact that this particular name is being used for a constant and not a variable.

6.8.1 Example program 1 - Calculating the circumference of a circle

Problem: Write a program that calculates the circumference of a circle, when supplied with the radius. The value of PI is defined as a constant.

Solution: A typical solution to the problem is shown in Code 8. The circumference is calculated using the formula: 2 x PI x r.

Note. It is good programming practice to include internal commentary in every program. The information makes the program more readable and also breaks the code listing into meaningful chunks.

The comment symbol is the single quote ('). Visual BASIC ignores all text that comes after the quote. Comments can also be embedded within the lines of code.

```
Option Explicit
Private Sub cmdCircle_Click()

'program circle

'10th February 2004
'Program by Fred Fink

'This program prompts the user to enter a value for the radius
'of a circle.  It then uses the formula: 2*PI*radius to
'calculate the circumference

Const PI As Single = 3.14159      'Declare constant PI and assigns value

Dim radius As Single
Dim circumference As Single

radius = InputBox("Please enter the radius of the circle")

circumference = 2 * PI * radius

Print "The circumference of circle with radius :";
radius; " is: "; circumference

End Sub
```

This file (Circle.txt), can be downloaded from the course web site.

Code 8

In this program an Input Box is used for input and a Print statement sends output direct to the form, see Figure 6.5 and Figure 6.6.

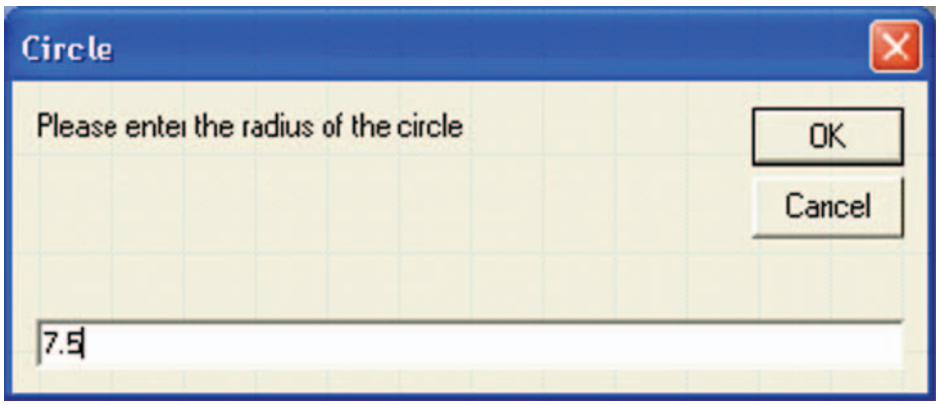


Figure 6.5:

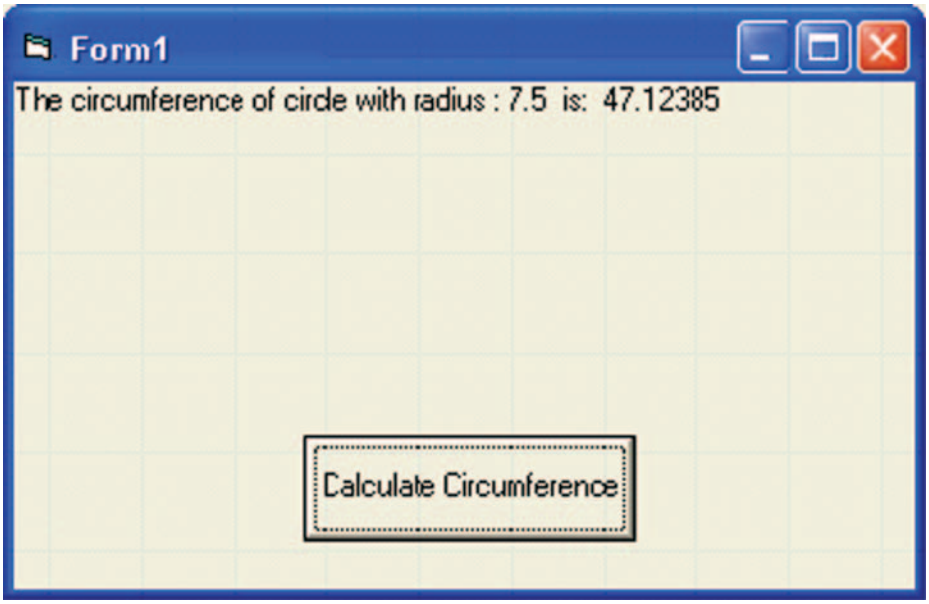


Figure 6.6:

Note that this method uses multiple windows. Input is via an inputBox and output is on the form itself, using the *print* command.

A VB project


30 min

1. create a new VB project.
2. create a form based on Figure 6.6, and name the button cmdCircle
3. enter the code for clicking the command button (you can cut and paste Code 8)
4. run the program to check it works correctly

6.8.2 Example program 2 - Use of a boolean variable

Problem: A program is required to display whether a number entered at the keyboard is greater than or equal to zero. The program will prompt the user to enter a number. It will then display a message, together with the value true or false, depending on the value of the number entered.

Solution:

The algorithm is shown below:

```
1. issue message to enter number
2. read the number typed in at the keyboard
3. set the boolean variable to true or false
       depending upon number >= or number < 0
4. display result message
```

The full Visual BASIC code for this program is shown in Code 9:

```
Private Sub cmdRun_Click()
```

```
'10th February 2004
'Program by Fred Fink

Dim number As Integer
Dim result As Boolean

' This is a program that will prompt the user for a number
' It will then check whether the number is less than or equal
' to zero or greater than 0
' A boolean variable will hold the result. A message saying
' true or false will be printed on the screen to say whether it is
' greater than zero
'Start of main program number = InputBox("Please enter a number")
result = number >= 0
Print number; "greater than or equal to zero = "; result
End Sub
```

Code 9

In this program the InputBox function is used to input the data, as before. The output is shown in Figure 6.7:

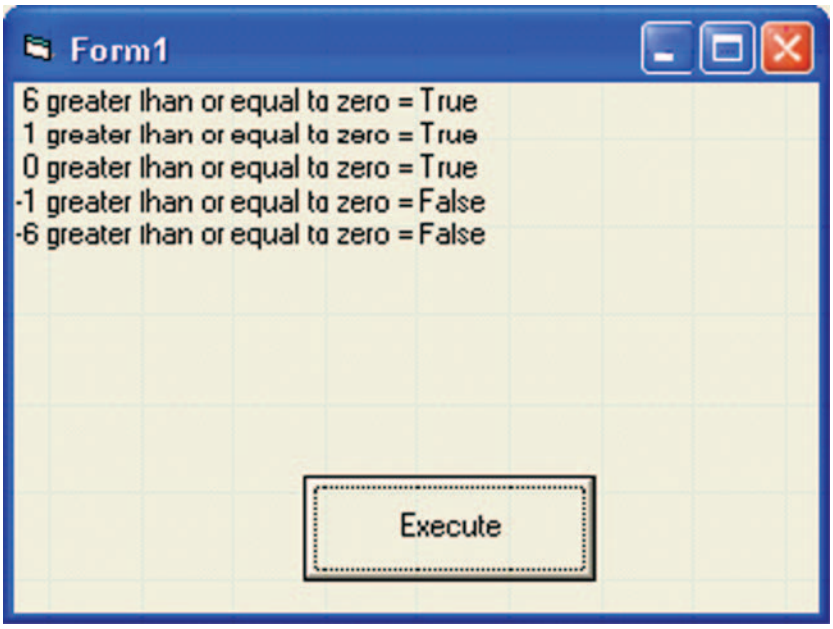


Figure 6.7:

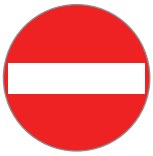
Boolean values are easy to use, and when used, they can often make a program more readable.

When defining a list of variables with the same type specifier, the variables can all be defined on the same line as the type specifier or separate type specifiers can be used for each variable. For example:

```
Dim number1 As integer, number2 As integer, sum As integer
```

is equivalent to

```
Dim number1 As integer
Dim number2 As integer
Dim sum As integer
```



However the statement:

```
Dim number1, number2, sum As integer
```

will assign *sum* as integer with *number1* and *number2* being assigned *variant*.

6.8.3 String variables and functions

A *string* is simply a collection of characters that are defined within quotes. The following are examples of string values:

"coffee"
"This program is not working"
"xyz"
"February 16,2004"
"Visual BASIC 6"
"Shrove Tuesday is pancake day"
"51"

The quotation marks are very important since they tell Visual BASIC that the enclosed characters are a *string*. Characters not enclosed in quotation marks are considered to be a numeric variable or some other part of the Visual BASIC language.

Note. The string value "51" does not equal the value 51. You might think that it's a value for an integer, but it's not. However there is a Visual BASIC function, VAL, that will convert string variables to values. The following statement:

```
NumericalValue = VAL("51")
```

will allow the variable `NumericalValue` to be assigned the integer value `51`.

Of course the reverse is also true. Given a value, it can be converted into a string using the Visual BASIC `STR$` function. The following statement:

```
StringValue = STR$(51)
```

will allow the variable `StringValue` to be assigned the string `"51"`

Two other useful functions are `CHR$` and `ASC`. The following examples show how they are used:

```
LetterValue = ASC("A")
```

will assign the value 65 to `LetterValue`. This is the ASCII code for the letter "A".

```
Letter$ = CHR$(66)
```

will assign the string "B" to `Letter$` since the ASCII code for "B" is 66.

Table 6.4 summarises the string functions:

Table 6.4:

Function	Usage	Examples	Result
`VAL`	Converts string to value	`Val("66")`	the integer 66
`STR$`	Converts value to string	`Str$(77)`	the string "77"
`ASC`	Converts string to ASCII value	`ASC("D")`	the ASCII value 68
`CHR$`	Converts ASCII value to string	`CHR$(68)`	the character "D"

String Functions

Online there is an interactivity which you should complete now.

6.8.4 Concatenation

Concatenation is simply joining string variables together to make longer strings.

The operator is the ampersand symbol (&). When two or more strings are combined the second strings are added directly to the end of the preceding string. The result is a longer string containing the full contents of both source strings.

The following example shows concatenation structure:

```
NewString = StringOne & StringTwo &                StringThree
```

Here `NewString` represents the variable that contains the result of the concatenation operation. `StringOne`, `StringTwo`, and `StringThree` all represent string variables.

Note that the ampersand must be preceded and followed by a space.

Example of concatenation

Example 1

The statement:

```
Print "Man" & "chester"
```

would produce the string `"Manchester"`

Example 2

```
String1 = "This is "
String2 = "con"
String3 = "cat"
String4 = "en"
String5 = "ation"

Print String1 & String2 & String3 & String4                & String5
```

will produce as output:

```
"This is concatenation"
```

6.9 Variables and scope

Variables have another important characteristic called **scope**. This determines which parts of a program are able to 'see' the variable and change its value.

The scope of a variable is determined not only by the type of declaration but also the declaration's location. For instance, the Dim keyword assumes different meanings in different parts of a form's code.

In Visual BASIC variables can be declared as shown in Figure 6.8:

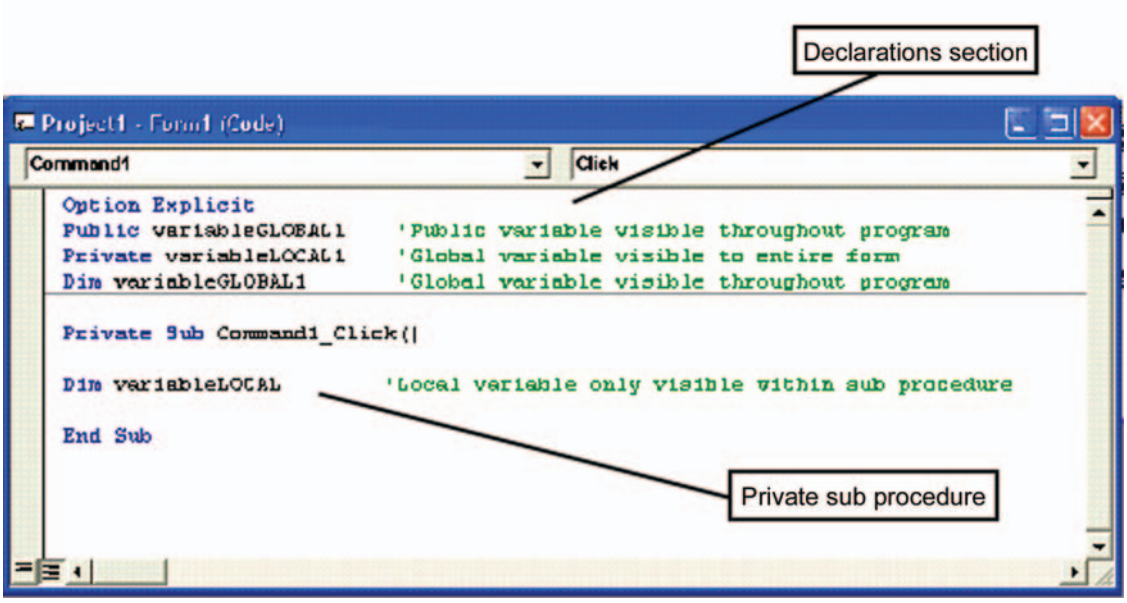


Figure 6.8:

1. If a variable is declared in the general declarations section with the `Dim` statement then it is **Global** to the entire program.
2. If declared within the sub procedure then the variable becomes Local to that sub procedure and will not be recognised elsewhere.

Note: VB also allows variables to be declared as Public or Private. A Public variable is Global and can be accessed from anywhere in the project. A Private variable can only be accessed from the form in which it was declared.

Consider the following program in Code 10:

```
Option Explicit
'Program to demonstrate global and local variables

Public TestVariable As Integer
Dim TestString As String

Private Sub ExecuteMain_Click()
  TestVariable = 55
  TestString = "Hello 1"
```

```
  Print "In main program Testvariable = "; TestVariable
  Print "In main program TestString = TestString"
End Sub


Private Sub ExecuteLocal_Click()
  Dim TestVariable As Integer
  Dim TestString As String
  Print
  Print "Call 1 gives TestVariable = "; TestVariable
  Print "Call 1 gives TestString = "; TestString
  Print
  TestVariable = 110
  TestString = "Hello 2"
  Print "Call 2 gives TestVariable = "; TestVariable
  Print "Call 2 gives TestString = "; TestString
End Sub
```

This file (Scope.txt), can be downloaded from the course web site.

Code 10

`TestVariable` and `TestString` are declared `Public`.

They are given values in sub procedure `ExecuteMain_Click()`.

They are re-declared in sub procedure `ExecuteLocal_Click()` and assigned new values.

Figure 6.9 shows the program output. The program is called twice by activating the two buttons.

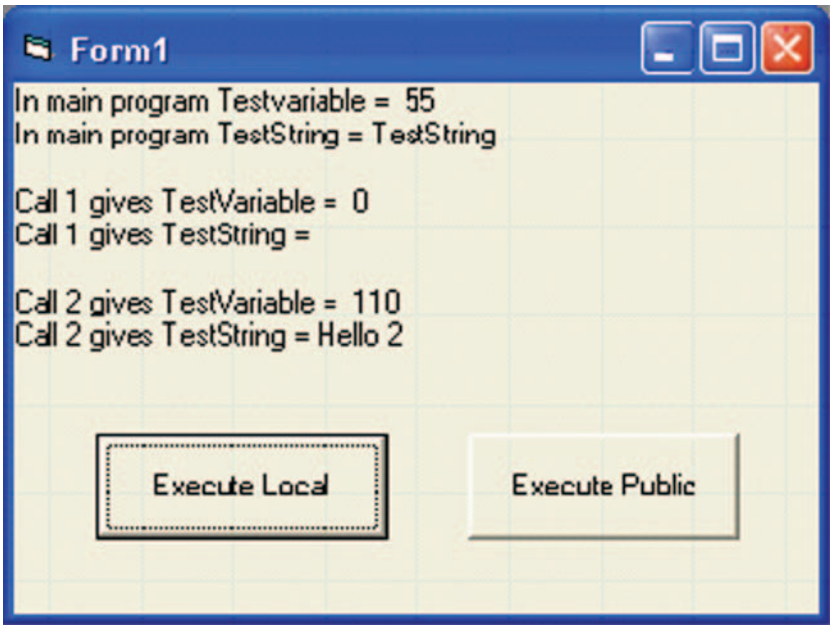


Figure 6.9:

Public gives the original values(that can change).

In call 1 local procedure resets the variables to 0 (since they have been re-declared).

Call 2 produces the new values from the local procedure.

Therefore, local variables always win!

Note: Extensive use of global variables is bad practice; the use of global variables should be kept to a minimum since their values can change from anywhere within a program as you have just seen.

Local variables on the other hand offer the following advantages:

1. side-effects caused by a subprogram altering the value of a variable used elsewhere in the program are greatly reduced
2. debugging is easier since access to variables is localised and so tracking errors can be faster
3. the transfer of procedures and functions from one program to another is simplified.

Sentence completion - global and local variables

On the Web is an interactivity. You should now complete this task.

6.9.1 Review Questions

Q9: In high level language code a variable declared outside a procedure is (choose one):

a) Local
b) Constant
c) Global
d) Functional

Q10: Which one of the following describes the relationship between main memory and program variables?

a) Name of variable identifies memory location
b) Data type defines how much memory is needed
c) Values of variables are held in main memory
d) All three statements above

Q11: Which one of the following describes a **local** variable?

a) Can be accessed anywhere in the program
b) They are hidden from other procedures and functions
c) Their values can easily be altered by mistake
d) The use of local variables is discouraged

Q12: The scope of a variable is an important aspect in programming. Scope is best described as:

a) The range of values the variable can cope with
b) The amount of memory required by the variable
c) The extent to which the variable can be 'seen' by the rest of the program

d) The number of times the variable can be used in the program

Q13: Which one of the following describes a boolean variable?

a) It holds a numerical value
b) It cannot be used in expressions
c) It is only used in looping structures
d) It can have only the values true or false

6.10 Operators

Assignment operator

The assignment operator in Visual BASIC is the equals (=) sign. The general form for the assignment operator is:

```
variable = expression
```

This statement says make left hand side equal to the right hand side. It makes the variable take on the same value as expression. You have already seen this used in previous programs.

For example:

```
circumference = 2 * PI * r
```

where the result of the expression `2* PI * r` is assigned to the variable `circumference`

Arithmetic operators

Table 6.5 list the main arithmetic operators in Visual BASIC:

Table 6.5:

Arithmetic operators	**Example**	**Result**
+ (add)	16 + 14	30
- (subtract)	27 - 9	18
/ (real division) (DIV)	27.83 / 3	9.27
* (multiply)	12.65 * 5	63.23
\ (integer division)	16 \ 7	2
mod (modulus operator)	25 mod 7	4
^ (raise to the power)	10 ^ 3	1000

It is important to remember that integer division gives you a whole-number answer only - any remainder is discarded. For example:

5 \ 2 gives 2 as the answer;
16 \ 5 gives 3 as the answer.

The modulus operator is used for division to obtain the remainder. The expression

```
x mod y
```

produces the remainder when `x` is divided by `y`

The `mod` operator can be applied to integers and also with `real`. For example:

5 mod 2 gives 1
8 mod 3 gives 2.
9 mod 3 gives 0 (zero).
7.1 mod 3.1 gives 1 (numbers rounded down)
10.8 mod 3.6 gives 3 (numbers rounded up)

Questions on Mod and Div

Q14: 7 Mod 2 =

a) 3.5
b) 3
c) 1
d) 49

Q15: 7/2 =

a) 3.5
b) 3
c) 1
d) 49

Q16: 7\2 =

a) 3.5
b) 3
c) 1
d) 49

6.10.1 Operator precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence. Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before other parts. Operations within parentheses are always performed before those outside. Within parentheses, however, normal operator precedence is maintained.

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

Arithmetic operators are evaluated in the following order of precedence:

Arithmetic operator precedence	
Parentheses ( )	1
Exponentiation (^)	2
Negation (-)	3
Multiplication and division (*, /)	4
Integer division (\)	5
Modulus arithmetic (Mod)	6
Addition and subtraction (+, -)	7

Operator Precedence

Online there is an interactivity which you should complete now.

Example 1 - Operator precedence: 4 + 5 * 2

Problem: What is the result of 4 + 5 * 2?

Solution: The multiplication is calculated first, then the addition since multiplication has a higher precedence than addition, e.g.

```
4 + 5 * 2 = 4 + 10 = 14
```

and so 4 + 5 * 2 is equal to 14.

Example 2 - Operator precedence: (4 + 5) * 2

Problem: What is the result of (4 + 5) * 2?

Solution: The expression in the brackets is calculated first, then the multiplication - the brackets have altered the order of precedence, e.g.

```
(4 + 5) * 2 = (9) * 2 = 18
```

and so (4 + 5) * 2 is equal to 18.

Example 3 - Operator precedence: 3 + (7 - 5)^2 * 4

Problem: What is the result of 3 + (7 - 5)^2 * 4?

Solution: The expression in brackets is evaluated first, then exponentiation, followed by multiplication then addition, i.e

```
3 + (7 - 5)^2 * 4 = 3 + 2^2 * 4 = 3 + 4 *
                  4 = 3 + 16 = 19
```

and so 3 + (7 - 5)^2 * 4 is equal to 19

Matching operations and results

On the Web is an interactivity. You should now complete this task.

6.11 Programming constructs

Structured programming is based on three constructs:

- sequence
- selection
- repetition

6.11.1 Sequence

This is the computer's basic mode of operation. This is how it's designed to work. A computer is designed to carry out an instruction and move automatically onto the next instruction.

A program is working in **sequence** when it

- begins at the beginning
- carries out each statement or instruction once and only once
- stops when it's reached the end

In other words, it misses nothing out (which happens under **selection**) and it does nothing more than once (which happens under **repetition**).

Sequence is the default mode of operation for a program: a program works in sequence unless it is explicitly told to do otherwise by means of one of the other programming constructs. Left to itself, it will always carry out a statement and proceed automatically to the next.

Example of simple sequence

The program examples you have seen so far follow simple sequences.

Try the following activity.

Problem: A program is written to ask the user to enter a number at the keyboard. The program then doubles it and displays the result on the screen.

```
1. request a number from the user
2. get the number typed in by the user
3. double the value of the number
4. display the new value on the screen
```

Sequential control of a program limits the type and range of problems that can be solved. Sequence is basic and essential, but limited. A program in simple sequence always does the same sort of thing, and it can't easily do things more than once.

We want programs that can carry out tests and make decisions. We want programs that can do things over and over again.

With the use of selection and repetition, the range of problems that can be solved becomes much larger. This is what we will look at next.

6.11.2 Selection

A fundamental task in any program is the decision of what to do next. Control constructs enable you to make decisions in your programs to determine when certain parts of the program will be executed.

In a similar manner to English you can make decisions using the if statement - for example, if it is raining then I will take my umbrella.

In this topic you will look at the following control constructs and how to use them to enable decisions to be made in programs.

- the `if` statement
- use of logical operators in decision-making
- the `if...else` statement
- nested `if` statements
- multi-way branching with `elseif`
- the `case` statement

6.11.3 Repetition

Repetition control structures are covered in the next topic, following a look at the details of selection control structures.

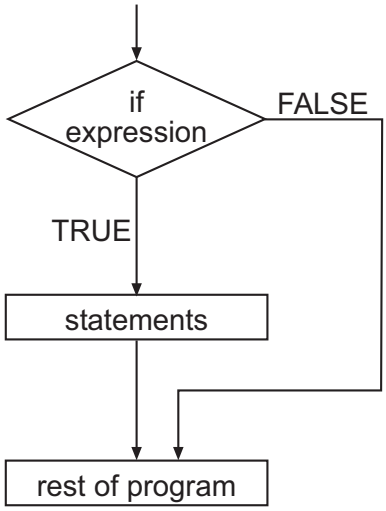
6.12 The IF Statement

The general decision making capability in Visual BASIC is provided by the `if` statement. The format of this statement is:

```
if condition then statement
```

or

```
if condition then statements end if
```



The program statement may be a single statement or may be a block of statements.

When the expression is true, the statements following are executed. When the expression is false, then none of the statements are executed and control passes to the remainder of the program

Example 1 - Using a simple IF statement

Problem: A program is required to ask the user for a number. If the number entered by the user has the same value as a value stored in a constant, then the program will display a suitable response.

The algorithm is shown below:

Solution:

```
1. request a number from the user
2. get number typed in at the keyboard
3. IF (the number typed in = the program constant) THEN
4. display "I like 99s"
5. display "Program End! So you don't like 99s "
```

The full Visual BASIC code for this program is as follows:

```
Option Explicit
Private Sub cmdExecute_Click()
  'program simple If_Then

  '13th February 2004
  'Program by Fred Fink

  'A program to show a simple if..then statement
  Const Ice As Integer = 99
  Dim userno As Integer

  userno = InputBox("Give me a number")
```

```
  If userno = Ice Then

    PicDisplay.Print "I like "; Ice; "s"

  End If

  PicDisplay.Print "Program End! "
End Sub
```

Code 11

In this program input is via an InputBox and output by means of the *PictureBox* function. Although a *PictureBox* is meant for graphics it can deal with text as well and this somewhat tidier than displaying text on a form.

The *PictureBox* property is set to the name *PicDisplay* in the properties window.

Program output is seen in Figure 6.10:

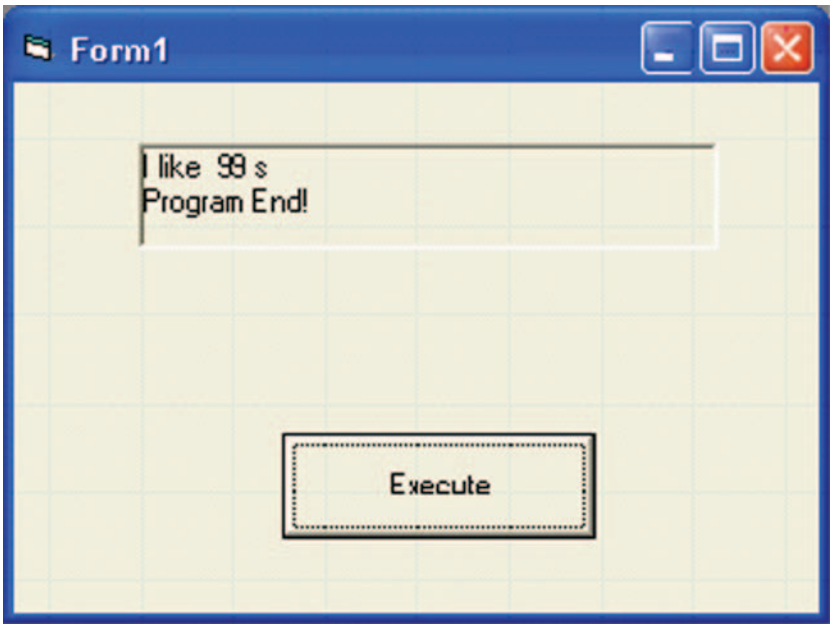


Figure 6.10:

Practical Examples using If .. Then

Several examples are given here that use comparison operators and `if` constructs within a program to solve a problem. Try writing the programs for yourself and experiment with making changes to the code to ensure you understand what is happening.

Example 1 - Calculating the area and circumference of a circle

Problem: Write a program that calculates the area and circumference of a circle. The calculations should only be performed if the radius is positive (i.e. a valid radius).

Solution: In the previous topic an example was given for calculating the circumference of a circle using a constant for the value of pi. This example could be modified to use an if statement to validate the radius i.e

```
if radius > 0 then
  do calculations
display results
```

Example 2 - Write a program to input numbers from the keyboard and to print them out. The program terminates when -1 is entered. A suitable message should be output.

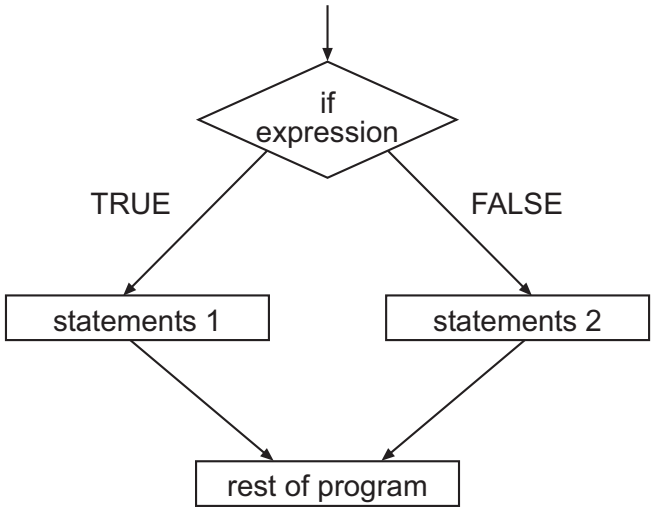
```
input number
if number > 0
  output number
end
terminate program
output message
```

6.13 The If.. Then.. Else Statement

So far we have shown you the use of the if..then statement when there is only one alternative i.e. if the expression is true then execute a single code fragment. If the expression is not true, you may want to execute a **different** code fragment. This is known as an If .. Then .. Else statement, and the general format is:

```
if condition then
  statements 1
else
  statements 2
End if
```

Again the statements may be single or they can be multiple statements.



If the expression is true, then the code fragment following the then (statements 1) will be executed. If the expression is false, the statements following the else (statements 2) will be executed.

Example - Single or double?

Problem: A program is written to prompt the user for a number. If the number entered by the user has a single digit, then the program will display an appropriate message telling the user that this is single-digit number. Otherwise it will output the message "double-digit number"

The algorithm is shown below:

Solution:

```
1.Ask for user to input number between 1 and 99
2.if the number < 10
3.display "single digit number"
4.else
5.display "double digit number"
```

The full Visual BASIC program is seen in Code 12

```
Option Explicit
Private Sub Command1_Click()

'program simple_If_Then_Else

'15th February 2004
'Program by Fred Fink

'This program will prompt the user to enter a number
'The program will test the number to see if it is
'less than 10 and print the message that it is a single-digit number.
'Otherwise it will output "double-integer number"

Dim testNumber As Integer

testNumber = InputBox("Please input a number ")

If testNumber < 10 Then
   PicDisplay.Print testNumber; "is a single digit number"
       Else
    PicDisplay.Print testNumber; "Is this double-digit number"
End If

End Sub

Private Sub cmd_Exit_Click()
End
End Sub
```

This file (IfThen.txt), can be downloaded from the course web site.

Code 12

Note that this program has an extra procedure - *Sub cmd_Exit_Click()*. You have probably found that Visual BASIC programs do not terminate by default. By adding an extra button on the form and programming this with '*End*' between the lines of code, this will terminate the program.

This will be used on all subsequent programs.

Sample output is shown in Figure 6.11.

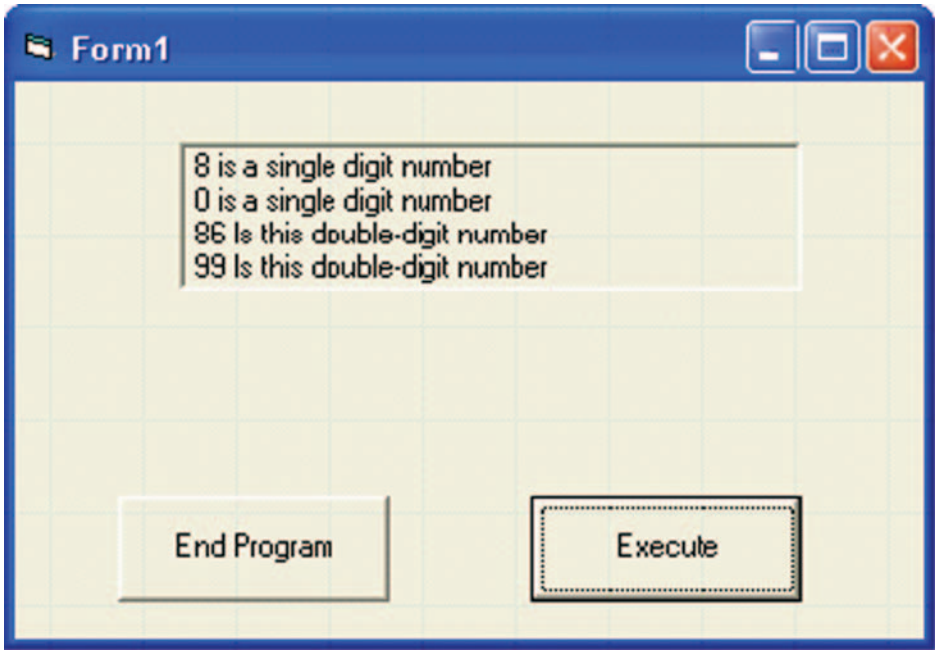


Figure 6.11:

Further examples

Example 1

Problem: What is the output of the following code?

```
Dim Number1 As Integer, Number2 As Integer

  Number1 = 10
  Number2 = 5
  if Number1 > Number2 then
    Print "Number1 is   bigger"
  else
    Print "Number2 is bigger"
  End if
```

Solution: This piece of code first declares two integer variables, `Number1` and `Number2`. `Number1` is assigned a value of 10, and `Number2` is assigned a value of 5.

The condition for the `if` statement is "is `Number1` greater than `Number2`"? If this is true then the statement "Number1 is bigger" is displayed. If the condition is false then the statement "Number2 is bigger" is displayed on the screen.

In this case the condition is true since `Number1` has a value of 10, which is greater than the value of 5 that has been assigned to `Number2` and so the output from this code is

```
Number1 is bigger
```

30 min

Which is bigger?

Write a program which prompts a user to type in two numbers - first one, then the other. The program responds by printing the bigger of the two. Use the code fragments in the examples above to help you, and make sure that the program has user-friendly prompts for the input and output.

30 min

Even or odd?

Write a program to test if a user entered number is even or odd.

In this program you will have to use the modulus operator (`mod`) to calculate the remainder when the value stored in the variable `number` is divided by 2. An even number has no remainder if divided by 2.

20 min

Calculating wages

Write a program to calculate the commission based wages of a computer salesman. His basic wage is £50 per week and he is expected to sell at least 10 computers. If he sells more than 10 computers, he receives an extra £5.50 per computer he sells after the 10th computer.

The basic algorithm could be drawn as shown in Figure 6.12

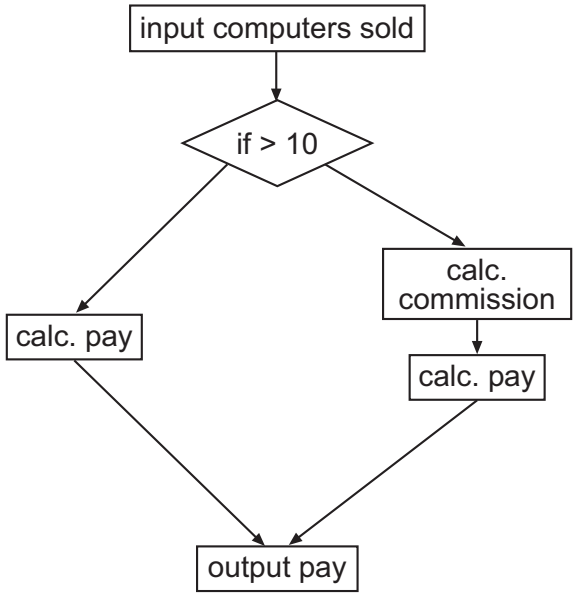


Figure 6.12

6.14 Comparison Operators

There are two kinds of comparison operators that allow comparison within Visual BASIC:

1. Relational,
2. Logical:
   These are very useful when implementing selection constructs.

6.14.1 Relational operators

Relational operators are used for testing of conditions. They are used to construct the 'expression' which is used in the `if` statement. The relational operators check whether two quantities are the same or whether there is a difference between them. Because of the inexact way floating point numbers are stored in computers you should *not* use the equality operator = between two real numbers. Even if two real numbers are printed to the screen as the same two numbers, this is no guarantee that they are the same internally.

The only way to test equality between real numbers A and B is to use the following expression:

```
If ABS(A - B) < small difference Then.....
```

The function `ABS` returns the absolute value of any number.`ABS(5.3)` gives 5, `ABS(-43)` gives 43

Depending upon how accurate real numbers are held on the computer small difference is usually of the order 0.0000001.

The relational operators are summarised in Table 6.6. These operations can be performed between most types of object, but they will be used most often to compare two integer values or two characters.

Table 6.6: Relational Operators

Symbol	Example	Meaning
=	`a = b`	`a` equal to `b`
<	`a < b`	`a` less than `b`
<=	`a <= b`	`a` less than or equal to `b`
>	`a > b`	`a` greater than `b`
>=	`a >= b`	`a` greater than or equal to `b`
<>	`a <> b`	`a` not equal to `b`

Example 1 - Using a Relational Operator

Problem: How can a comparison be made between two variables, `Num1` and `Num2` to find out if `Num1` is greater than or equal to `Num2`?

Solution: The comparison can be made using the relational operator >=, e.g.

```
Num1 >= Num2
```

This comparison could be reversed to check if `Num2` is less than `Num1`, e.g.

```
Num2 <= Num1
```

Example 2 - Using a relational operator in an `if` statement

Problem: Construct an if statement that will display a message to the screen if two variables, `Val1` and `Val2`, are equal.

Solution: A typical solution is shown here.

```
Dim Val1 As Integer, Val2 As integer
...
  if Val1 = Val2 then
    Print "Val1 and Val2 are equal"
  end if
```

Sentence completion - relational operators

On the Web is an interactivity. You should now complete this task.

6.14.2 Logical Operators

Logical operators test conditions as either being *true* or *false*. In computer programming they are referred to as Boolean operators, named after George Boole, a mathematician and logician.

6.14.3 Logical AND

Logical `AND` means that an expression is true only if both the part before the '`AND`' and the part after the '`AND`' are true. Both must evaluate to true . So...

```
if (number < 0) and (number > 10) then
```

the expression will always evaluate to `false`. Why? Because a number cannot be less than zero and greater than 10 at the same time, and both must be `true` for the '`AND`' statement to produce a `true` result.

Use of the logical '`AND`' operator can be seen in the expression below:

```
if (Vone = 6) and (Vtwo = 6) then
  statement 1
    else
  statement 2
End if
```

This code means that when both `Vone` and `Vtwo` are equal to 6 then `statement 1` is executed. If either or both of the conditions is `false` then `statement 1` is ignored and execution passes onto `statement 2`.

Online there is an interactivity which you should attempt now.

Example 1 - Passwords

Problem: A program is written which asks the user to enter a password and to confirm it by entering it again. If both words are equal to a program constant then the program

will display a suitable comment. If not, or only one instance of the password is entered the user will be informed.

The algorithm is shown below:

Solution:

```
1. request 1st password from the user
2. request 2nd password from the user
3. if (the first password = constant) and (the second password = constant) then
4. display the word entered both times
5. else
6. display the two words and that they are different"
```

The full Visual BASIC code for this program is as follows:

```
Option Explicit
Private Sub Command1_Click()

  'program using the AND operator

  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter two words
  'and compare them to a string constant

  Dim Word1 As String
  Dim Word2 As String

  Const pass As String = "Daffodil"

  Word1 = InputBox("Please input password ")
  Word2 = InputBox("Password again, please")

  If (Word1 = pass And Word2 = pass) Then
    PicDisplay.Print "Both words "; pass; " are identical"
       Else
    PicDisplay.Print Word1; " and "; Word2; " are not identical"
  End If

End Sub


Private Sub cmd_Exit_Click()
End
End Sub
```

This file (Password.txt), can be downloaded from the course web site.

Code 13

Program output is seen in Figure 6.13

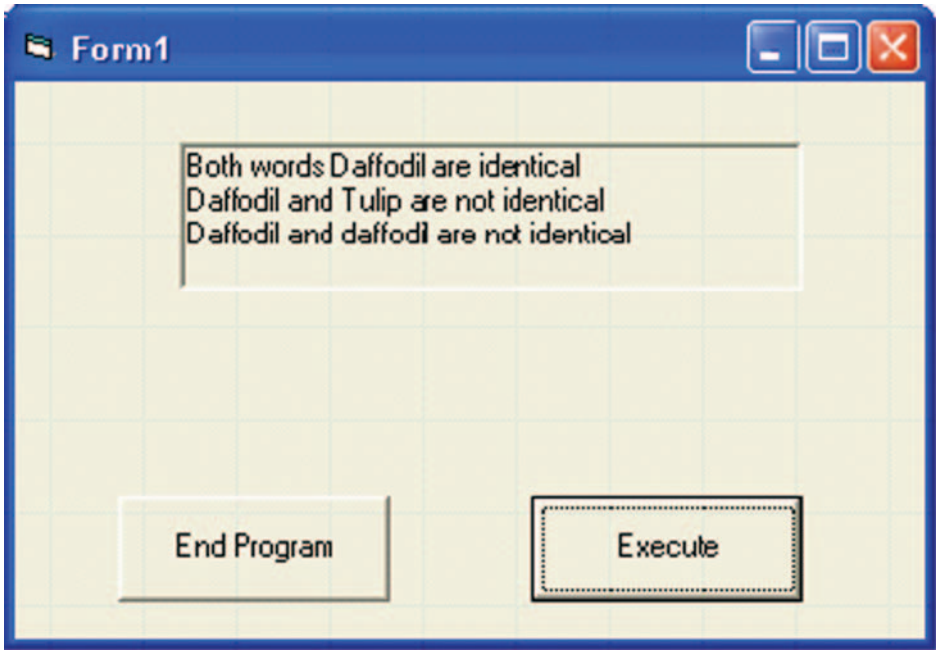


Figure 6.13:

Notice from the third entry that the input is case sensitive.

This could be resolved using the UCase or LCase pre-defined functions.

Using the logical `AND` operator in an `if` statement

Construct an `if` statement that displays a message if both the variables, `number1` and `number2` are greater than 10.

20 min

Validating numeric input and program testing

Use fragments of code you have just seen in this section to help you write a program which will accept a value from the keyboard and print 'yes' if it is **positive** and **even**. Test your program with positive and negative numbers, odd and even numbers and make sure it passes all the tests. Make a tabulated list of the numbers you use and the results the program gives for each input. Use both positive and negative numbers, not neglecting zero.

6.14.4 The Logical OR (Inclusive)

If either the expression before the `OR` or after the `OR` is `true`, then the whole expression is true. This also means that if both the expressions are true, the `OR` expression will evaluate to `true`. So or means 'either one, or the other or both'. Hence the term *inclusive*.

Use of the logical OR operator is shown below:

```
if (Vone = 6) or (Vtwo = 6) then
  statement 1
else
```

```
  statement 2
end if
```

This code means that when either `Vone` or `Vtwo` is equal to 6 then `statement 1` is executed. If both of the conditions are false then `statement 1` is ignored and execution passes onto `statement 2`.

Online there is an interactivity which you should attempt now.

Example 1 - Wash the car?

Problem: A program is written to prompt the user to enter two conditions - a temperature and a weather forecast. If either or both of the conditions are met then the program will display a suitable message to go and wash the car. If no conditions are met the user will be informed that washing the car is not a good idea. The conditions are compared to program constants.

The algorithm is shown below:

Solution:

```
1. request a temperature from the user
2. request weather condition from user
3. if (the first input >= 15) or (the second input = "Sunny") then
4.   display "Wash the car!"
5. else
6.  display "Not a good idea!"
```

The Visual BASIC code for this program is shown in Code 14.

```
Option Explicit

Private Sub Command1_Click()

  'program using the OR operator


  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter a value and
  'a condition

  Dim iTemp As Integer
  Dim weather As String

  iTemp = InputBox("Please input today's temperature ")
  weather = InputBox("Please input Sunny or Rainy")

  If (iTemp >= 15 Or weather = "Sunny") Then
```

```
        PicDisplay.Print iTemp; "deg "; " and "; weather; "! Wash the car"
           Else
        PicDisplay.Print iTemp; "deg "; " and "; weather; "! No way!"
      End If

    End Sub


    Private Sub cmd_Exit_Click()
    End
    End Sub
```

This file (WashCar.txt), can be downloaded from the course web site.

Code 14

The program output is shown in Figure 6.15.

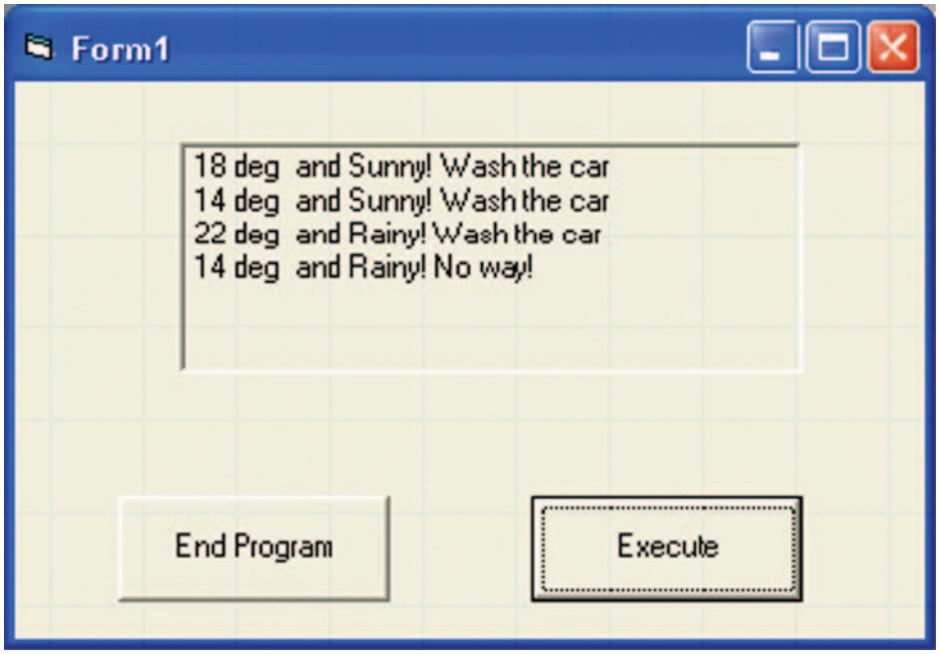


Figure 6.14:

6.14.5 Logical NOT

Use of the logical `NOT` operator is shown below:

```
if (Vone = 6) and not (Vtwo = 6) then
   statement 1
end if
```

In this example if `Vone` is equal to 6 and `Vtwo` is *not equal* to 6 then `statement 1` is executed. The `not` operator inverts true/false values e.g. if the value is `true` then `not(true) = false`.

Example The '*Word1 and Word2*' program could easily be modified to include the NOT operator. Rewrite the code and run the program

6.14.6 Review Questions

Q17: Operators are used within expressions to assign values to variables and perform calculations. Which one of the following operators has the highest precedence in Visual BASIC?

a) +
b) AND
c) /
d) NOT

Q18: What would be the result of evaluating the following expression 3 + 4 * (6 - 4)?

a) 11
b) 14
c) 23
d) 38

Q19: Which one of the following describes **sequencing**?

a) Alteration in the flow of control based upon the test of a condition
b) The repetitive execution of a sequence of instructions
c) The execution of program statements in order, from beginning to end
d) Exiting a loop structure after a sequence of instructions

Q20: Which one of the following expressions represents the logical AND operator?

a) Requires only one condition to be tested
b) The expression is TRUE when both conditions being tested are TRUE
c) The expression is TRUE if either the conditions being tested are TRUE
d) Can be used to test for values outwith a given range

Q21: Which one of the following is NOT a control structure in programming?

a) Sequence
b) Repetition
c) Assignment
d) Selection

Outwith range

When does the following statement evaluate to be TRUE?

```
(number < 0) or (number > 9)
```

6.15 Other Forms of If Statement

Other more complex variations on the If Statement are possible; these include:

A) Nested If constructs

B) If .. Then .. Elseif constructs These can be difficult to get right, and are usually best avoided by using either:

   a) a list of If statements, or

   b) the Select Case construct.

However, nested Ifs and If .. Then .. Elseif are included here for completeness, but can be omitted if desired.

6.15.1 Nested IF Statements (optional)

If a condition has to be tested that depends on whether another condition is already `True` (such as "If it's 6:30 p.m. and if I'm logged on to SCHOLAR" then.....), nested If statements can be used.

A nested If statement is one that's enclosed within another If statement.

The format for a nested If statement is as follows:

```
If condition Then
  If another_condition Then
    statement
  Else
    another statement
  End If
End If
```

`if` statements can be nested, but care should always be taken to ensure that the `else` statement is associated with the correct `if`.

Nested IF statements can be avoided by using the Select..Case statement, which is described in a later topic.

The following example exemplifies the nested *If* statement:

Example 1 - Testing for range and the number of digits in a number

Problem: Describe the operation of the following program as seen in Code 15:

```
Option Explicit

Private Sub Command1_Click()

  'program nested_IF

  '15th February 2004
```

```
'Program by Fred Fink

'This program will prompt the user to enter a number between 1
'and 99
'The program will test the number to see if it is
'less than 10 and print the message that it is a single-digit
'number.
'If it is > 9 it will output "double-integer number"
'otherwise "out of range"

Dim testNumber As Integer

testNumber = InputBox("Please input a number between 1 and 99")

If (testNumber >= 0) And (testNumber <= 99) Then
  If testNumber < 10 Then
    PicDisplay.Print testNumber; "is a single digit number"
    Else
       PicDisplay.Print testNumber; "is a double-digit number"
  End If
    Else
    PicDisplay.Print testNumber; "is out of range"
End If

End Sub

Private Sub cmd_endProgram_Click()
End
End Sub
```

This file (NestedIF.txt), can be downloaded from the course web site.

Code 15

Solution: One variable, `testNumber`, is declared as an `integer`. The user is prompted to enter a number between 1 and 99 and the value entered is stored in the variable `number`. If the value is not within the specified range (greater than 0 and less than 100), then the `else` part of the outer `if` statement is executed and the message *Number is out of range* is displayed on the screen and the program is finished.

On the other hand, if the value is within range, then the first part of the outer `if` statement is executed which contains several lines of code between `if` and `End If`. Within this block of code is another `if` statement which tests if the number is less than 10, in which case the message *single digit number* is displayed on the screen, otherwise (else), the message *double digit number* is displayed on the screen.

The program output is shown in Figure 6.15

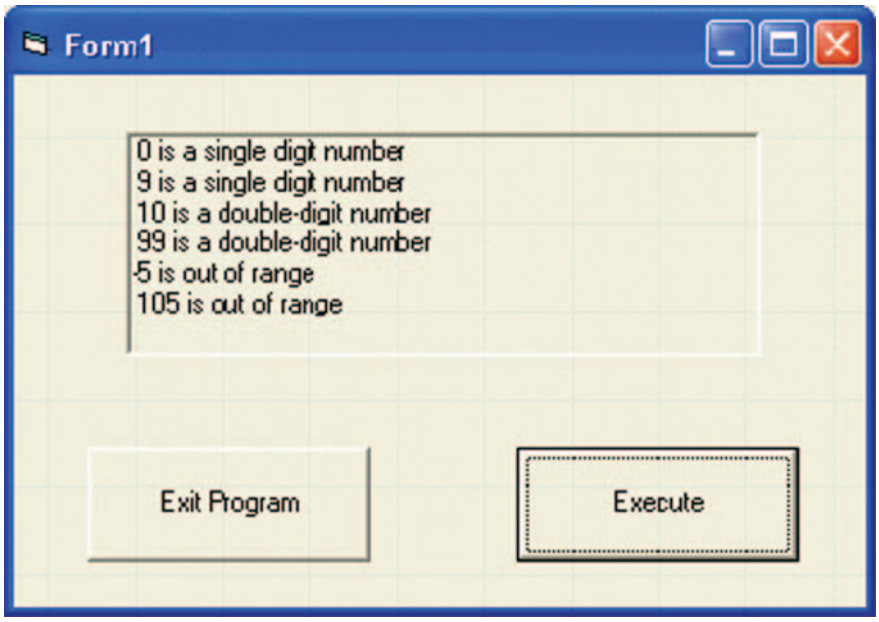


Figure 6.15:

Please take time to follow this explanation through. If you can see the 'how' and 'why' of it, you will be able to program `if` statements - nested or plain - without any worries

6.15.2 If...Then...ElseIf (optional)

Nested `if` statements can be fairly complex to follow and can give rise to what is known as '*spaghetti code*'. It is usually better to use Select..Case, as described in Section 6.16

The `ElseIf` statement can simplify expressions to a certain extent. The structure of the statement is:

```
if expression1 then
  statements 1
elseif expression2 then
  statements 2
elseif expression3 then
  statements 3
else
  statements 4
```

These expressions are evaluated in order, if any expression is true, the statement associated with it is executed and this terminates the whole chain. As many decisions as desired may be included within the chain. As always, the code for each statement is either a single statement or a group of statements.

Consider the situation where you have two boolean variables, `booly1` and `booly2`, and you need to take actions for each of the possible combinations of true and false.

Using nested `if` statements would produce:

```
If booly1 = True Then
  If booly2 = true Then
```

```
    statements for true/true
  Else
    statements for true/false
  End If
Else
  If booly2 = false Then
    statements for false/true
  Else
    statements for false/false
  End If
End If
```

Using `ElseIF` will produce the following, more compact code:

```
If booly1 And booly2 Then
  statements for true/true
ElseIf booly1 And Not booly2 Then
  statements for true/false
ElseIf Not booly1 And booly2 Then
  statements for false/true
Else
  statements for false/false
End If
```

The following examples will show the `If..EndIf` statement in use:

Example 1 - Using an elseif statement to calculate the number of digits in a number

Problem: How can an `else if` statement be used to find out whether an integer number is within a certain range (a positive number) and how many digits it contains?

Solution: A typical solution to this problem is shown here.

```
if number >= 1000 then
  n_digits = 4
elseif number >= 100 then
  n_digits = 3
elseif number >= 10 then
  n_digits = 2
elseif number >= 0 then
  n_digits = 1
else
  print("Value out of range")
```

The above code shows an example of how this could be achieved.

- You start testing if the variable `number` is greater than or equal to 1000, in which case there are 4 digits (you must be assuming that the variable `number` is not

greater than 999) and the variable `n_digits` is set to 4. If it is not greater than or equal to 1000, then you move onto the next branch of the `if` statement.

- The next condition tests the variable `number` to see if it is greater than or equal to 100. If this is true then you know the variable `number` has a value between 100 and 999, and so it must have 3 digits, and the variable `n_digits` is set to 3. Again, if this condition is not fulfilled, then you move onto the next branch of the `if` statement.
- The next condition tests the variable `number` to see if it is greater than or equal to 10. If this is true then you know the variable `number` has a value between 10 and 99, and so it must have 2 digits, and the variable `n_digits` is set to 2. Again, if this condition is not fulfilled, then you move onto the next branch of the `if` statement.
- The next condition tests the variable `number` to see if it greater than or equal to 0. If this is true then you know that the variable number has a value between 0 and 9, and so it must have one digit. The variable `n_digits` is set to 1.
- The next branch of the `if` statement is not an `elseif` branch, but an `else` branch, and so does not have a condition but is the branch that is executed if none of the other conditions are met. Execution of this branch of the `if` statement results in the message `Value out of range` being displayed on the screen.

Example 2 - Grades and Marks

Problem: Write a program using the `ElseIf` construct for a user to input a test result and output the corresponding grade. Test scores range from 0 to 100 and the grades from "A" being the highest to "E" the lowest.

Solution

Use the following algorithm:

```
1 ask user to input a mark between 0 and 100
2 If mark (>=0 and < 30) then grade "E"
3   elseIf mark(>=30 and <49) then grade "D"
4   elseif mark(>=49 and < 65) then grade "C"
5   elseif mark(>=65 and < 85) then grade "B"
6   elseif mark(>=85 and <=100) then grade "A"
7   else number out of range
8 end if
9  If mark (>=0 and <=100)
10   display grade
11 end if
```

The full Visual BASIC program is shown in Code 16:

```
Option Explicit
Private Sub Command1_Click()
  'Marks and grades program
  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter a mark and
  'a grade will be output

  Dim iMark As Integer
  Dim sGrade As String

  iMark = InputBox("Please input a test mark ")

  If (iMark >= 0) And (iMark < 30) Then
    sGrade = "E"
    ElseIf (iMark >= 30) And (iMark < 50) Then
    sGrade = "D"
    ElseIf (iMark >= 50) And (iMark < 65) Then
    sGrade = "C"
    ElseIf (iMark >= 65) And (iMark < 85) Then
    sGrade = "B"
    ElseIf (iMark >= 85) And (iMark <= 100) Then
    sGrade = "A"
    Else
    PicDisplay.Print iMark; " is out of range! Try again."
  End If

  If (iMark >= 0) And (iMark <= 100) Then
    PicDisplay.Print iMark; " = grade "; sGrade
  End If
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (Grades.txt), can be downloaded from the course web site.

Code 16

Figure 6.16 shows the program output:

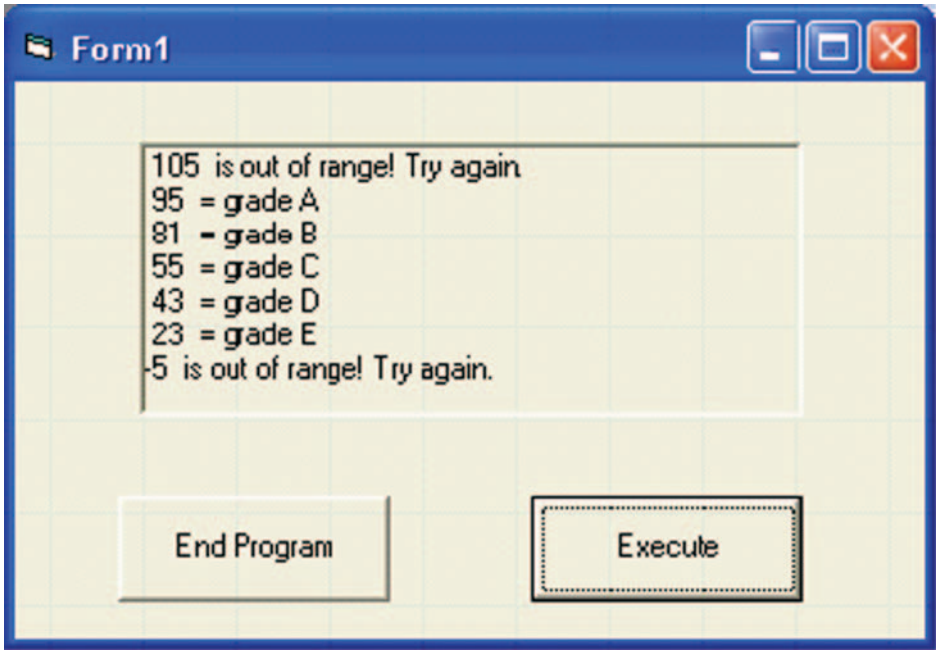


Figure 6.16:

20 min

Calculating the number of digits in a number

Write a program which will calculate the number of digits in a number. The error messages should be helpful and user-friendly. The program should deal correctly with numbers having four, or more, digits by giving the error message *Value out of range. Maximum of three digits numbers, please.*

20 min

Calculating Leap Years

A leap year is a year which is exactly divisible by 4, unless it is exactly divisible by 100 in which case it is only a leap year if it is exactly divisible by 400.

Write a program which prompts the user for a year and then displays if it is a leap year or not.

Note: Use `else if` in your code. You will also need to be careful that you do the tests for leap years and leap centuries in the correct order. If you test for leap years first, then 1900 is divisible by 4, but as it is not a leap century it will not be a leap year either.

6.16 The Select Case Statement

To implement **multi-way decisions** the `case` statement provides a more concise and elegant representation than multiple `else if` and `nested if` statements, which can get very difficult to follow.

The `case` statement is particularly useful when selection is based on a single variable or a simple expression. This is called the *case selector*. Note that the case selector must be an ordinal data type i.e. one whose values can be listed, such as `integer`, `string` or `boolean`.

The structure of the `Case` statement is:

```
Select case expression

Case value1
  Block of one or more statements
Case value2
  Block of one or more statements
Case value3
  Block of one or more statements
Case Else
  Final block of one or more statements

End Select
```

The expression is compared with each of the case values in turn. If there is a match, the relevant statements are executed. If there is no match, the `Else` statements are executed.

Several worked solutions to problems are given here - make sure you understand what is going on in these examples. Run these programs for yourself and experiment with making changes to the code to ensure you understand what is happening.

6.16.1 Select Case Example 1

Example 1: Write a program to input an integer between 1 and 7 and the day of the week will be output.

Solution

Use the following algorithm:

```
1 ask user to input number between 1 and 7 inclusive
2 select case number
3 case is = 1 output "Sunday"
4 case is = 2 output "Monday"
5 case is = 3 output "Tuesday"
6 case is = 4 output "Wednesday"
7 case is = 5 output "Thursday"
8 case is = 6 output "Friday"
9 case else output "Saturday"
```

```
10 end select
```

Note the use of keyword `is`. Whenever a condition is expressed within a statement the keyword `is` is used to impose the condition.

The full Visual BASIC program is shown in Code 17.

```
Option Explicit
Private Sub Command1_Click()
  'Use of the Case Select statement(1)
  '15th February 2004
  'Program by Fred Fink

  'This program will prompt the user to enter a number
  'and the day will be output

  Dim iDay As Integer

  iDay = InputBox("Please input a number between 1 and 7 ")

  Select Case iDay
    Case Is = 1
    PicDisplay.Print iDay; " =  Sunday"
    Case Is = 2
    PicDisplay.Print iDay; " =  Monday"
    Case Is = 3
    PicDisplay.Print iDay; " =  Tuesday"
    Case Is = 4
    PicDisplay.Print iDay; " =  Wednesday"
    Case Is = 5
    PicDisplay.Print iDay; " =  Thursday"
    Case Is = 6
    PicDisplay.Print iDay; " =  Friday"
    Case Else
    PicDisplay.Print iDay; " =  Saturday"
  End Select
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (Case1.txt), can be downloaded from the course web site.

Code 17

The program output is shown in Figure 6.17

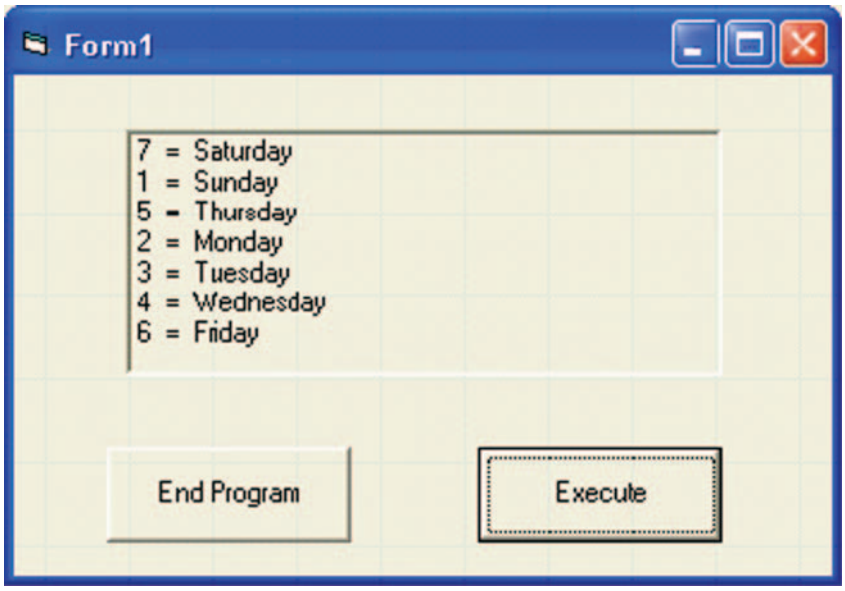


Figure 6.17:

6.16.2 Select Case Example 2

Example 2 - Using the `Select Case` statement with a range of values

Problem: Rewrite the previous `If..Then..ElseIf` program, Code 16, using `Select Case` Statement

Solution

Here the keyword `To` is used to represent values within a range.

Use the following algorithm:

```
1 ask user to input a mark between 0 and 100
   2 select case mark
3 case 85 to 100 output Grade 'A'
4 case 65 to 84 output Grade 'B'
5 case 50 to 64 output Grade 'C'
6 case 30 to 49 output Grade 'D'
7 case 0 to 29 output Grade 'E'
8 case else output "mark outside range! Try again"
9 end select
```

The full Visual BASIC program is shown in Code 18

```
Option Explicit
Private Sub Command1_Click()
'Use of the Case Select statement
'15th February 2004
'Program by Fred Fink

'This program will prompt the user to enter a date
'and the month will be output
```

```
Dim iMark As Integer

iMark = InputBox("Please input a test mark ")

Select Case iMark
    Case 85 To 100
    PicDisplay.Print iMark; " = grade A "
    Case 65 To 84
    PicDisplay.Print iMark; " = grade B "
    Case 50 To 64
    PicDisplay.Print iMark; " = grade C "
    Case 30 To 49
    PicDisplay.Print iMark; " = grade D "
    Case 0 To 29
    PicDisplay.Print iMark; " = grade E "
    Case Else
    PicDisplay.Print iMark; " is out of range! Try again."
End Select
End Sub

Private Sub cmd_Exit_Click()
End
End Sub
```

This file (Case2.txt), can be downloaded from the course web site.

Code 18

Program output is shown in Figure 6.18:

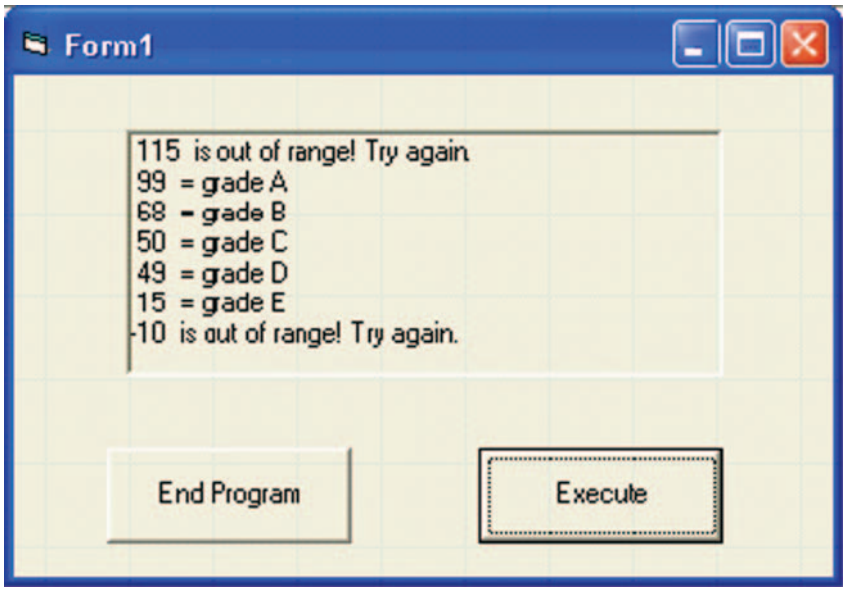


Figure 6.18:

6.16.3 Select Case Example 3

Example 3 - Extending the testing for a weekend or weekday using text Input

Problem: Write a program to test whether a `string` variable `Day` holds text that represents a week day or a weekend. Assume input can be in either lower or upper case.

Solution: The statement in the previous example, Code 17, could be extended to give the following program Code 19:

```
Option Explicit
Private Sub Command1_Click()

  'program use of case statement (3)
  '16th February 2004
  'Program by Fred Fink

  'To show the use of case with upper or lower case input

  Dim Day As String
  Dim FirstLetter As String

  Day = InputBox("Please enter the day")
  FirstLetter = Left$(Day, 1)     'Pick off first character

  Select Case FirstLetter
    Case "s", "S"
    PicDisplay.Print Day; " is the weekend; "
    Case "m", "M"
    PicDisplay.Print Day; " is a weekday "
    Case "t", "T"
    PicDisplay.Print Day; " is a weekday "
    Case "w", "W"
    PicDisplay.Print Day; " is a weekday "
    Case "f", "F"
    PicDisplay.Print Day; " is a weekday "
    Case Else
    PicDisplay.Print Day; " is not a day! "
  End Select

End Sub

Private Sub cmd_Exit_Click()
End
End Sub
```

This file (Case3.txt), can be downloaded from the course web site.

Code 19

Program output is shown in Figure 6.19:

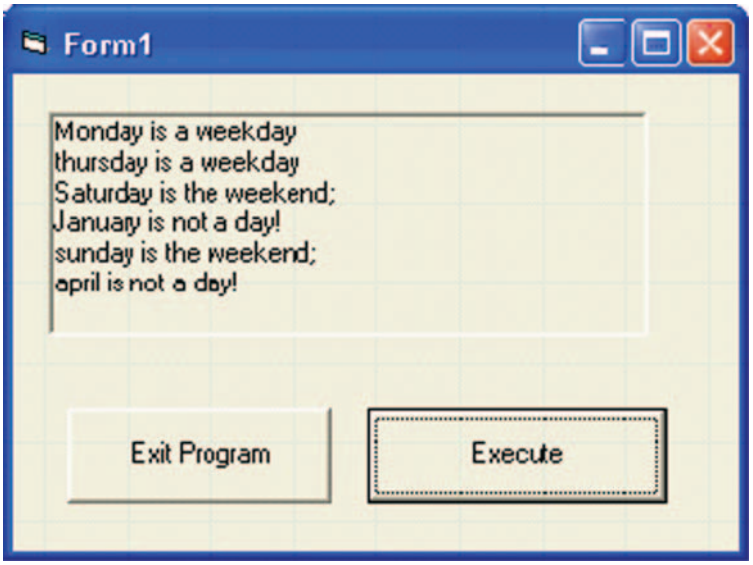


Figure 6.19:

It is really important that you study this example carefully. You will probably use the `case` statement a lot.

This program uses the `case` statement to check whether the letter typed in by the user matches `s` (Saturday and Sunday), `m` (Monday), `t` (Tuesday and Thursday), `w` (Wednesday) or `f` (Friday). If the match is with `s` it can only be a weekend. All other matches must be week days.

Note: Here you are comparing with the single character and not the entire string called Day so single quotes are required to show that you are using a character.

In order to test the first character of the string `Day` the Visual BASIC in-built function `Left$` is used.

For example `Left$(Monday, 1)` would produce the character `"M"`

6.16.4 Select..Case Summary

Sentence completion - `case` statement

On the Web is an interactivity. You should now complete this task.

40 min

Musical Notes

A, B, C, D, E, F and G are valid musical notes.

Write a program that allows the user to input a character and uses a `CASE` statement to test and display if the character is a valid musical note.

6.17 Summary

- an Input Box or text box can be used to get user input to a program;
- output can be directed to a Message box or to the form;
- output can be formatted using TAB and SPC functions;
- variables must be declared before use: name, and type;
- undeclared variant type should be avoided;
- variable types include integer, real (single or double), Boolean, string;
- string concatenation is implemented using the & character;
- the scope of a variable defines where it can be accessed from;
- wherever possible, local variables should be used rather than global variables;
- operator precedence must be considered carefully when evaluating complex expressions;
- selection constructs are implemented using If.. Then and If .. Then .. Else statements;
- nested Ifs and ElseIfs are best avoided using Select Case.

6.18 End of topic test

An online assessment is provided to help you review this topic.

Topic 7

High Level Language Constructs 2

Contents

Prerequisite knowledge

Before studying this topic you should be able to describe and use the following constructs in pseudocode and a suitable high level language:

- *fixed loops;*
- *conditional loops using simple and complex conditions;*
- *nested loops;*
- *1-D arrays.*

Learning Objectives

After completing this topic, you should be able to:

- *declare a 1-dimensional array;*
- *access array elements;*
- *manipulate data within arrays using iterative structures;*
- *format output.*

Revision

Q1: An array is described as a structured data type. This means that:

a) Data items are all in order
b) Data items will take up a lot of computer memory
c) Data items of the same type are grouped together
d) None of the above

Q2: A 1-D string array called Days(0-6) hold the days of the week. The 4th array element is assigned the value Wednesday. The correct statement for this is:

a) Days(4) = "Wednesday"
b) Days(3) = Wednesday
c) Days(4) = Wednesday
d) Days(3) = "Wednesday"

Q3: An array called List(5) contains the integers 1 to 6 in sequence. If the 2nd and 4th elements are now assigned the values 8 and 9 respectively, the array List will now contain:

a) 1, 2, 3, 8, 5, 9
b) 1, 2, 8, 4, 9, 6
c) 1, 8, 3, 9, 5, 6
d) 8, 2, 9, 4, 5, 6

7.1 Introduction

An important aspect of this topic is the 1-dimensional array. Declaring and initialising arrays are introduced together with how data is manipulated within arrays using various looping structures. Each sub topic has working solutions to the example programs, providing a suitable environment for building confidence in writing programs before the final topic, dealing with standard algorithms, is covered.

7.2 Fixed loops

The aim of this topic is to consolidate your knowledge of *iterative* structures, or loops. Iteration simply means repetition, which in the context of programming is the execution of blocks of code many times over.

Iteration is a fundamental part of almost every program and is one of the most useful features of programming. You do not want a computer to produce one payslip, but many payslips; to add up just two numbers but thousands of numbers; to put in order just two items but thousands of items.

There are three different looping constructs you can use in Visual BASIC:

1. For...Next loop
2. Do...While loop
3. Do...Until loop

Most loops have the following characteristics in common:

- initialisation
- a condition which evaluates either to TRUE or FALSE
- a counter that increments or decrements by discrete values.

For .. Next loops are examples of **fixed loops**, and we will study them first.

DO loops are **conditional loops** and we will study them later.

7.2.1 The For..Next loop

When *incrementing*, the general form of the For .. Next loop is:

```
For counter = initial value To final value Step value

    statements

Next counter
```

When *decrementing*, the general form is:

```
For counter = initial value To final value Step -ve value

    statements;

Next counter
```

Note:

1. The *initialisation* statement is carried out only once when the loop is first entered i.e. initialise counter to *initial value*
2. The *condition* is tested before each run through the body of the loop. The first test is immediately after initialisation, so if the test fails the statements in the body are not run. An incrementing loop terminates when *counter* $>$ *final*, while a decrementing loop terminates when *counter* $<$ *final*
3. An increment or decrement of the *counter* variable is executed after the loop body and before the next test. The value of the counter is incremented or decremented by the *step value*.

4. the value of *counter* must not be changed in any statements within the body of the loop
5. changing the value of *final* within the loop will have no *effect* on how many times the loop is executed
6. after the loop has terminated, the value of *counter* is *undefined*
7. counter may be any *ordinal* type (usually an integer)

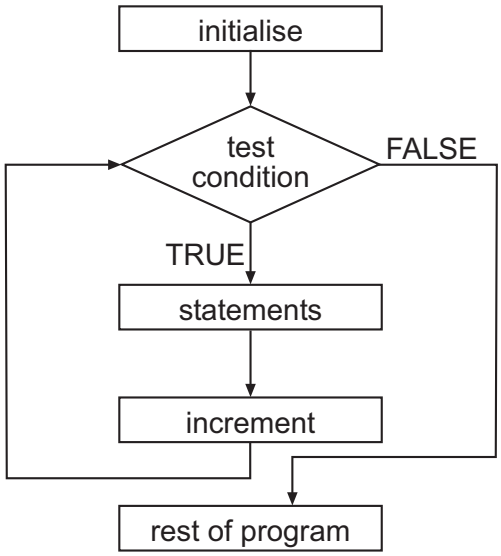


Figure 7.1:

The initial and final states may be *constants, variables* or *expressions* e.g.

```
for counter = (min+7) to (max-5)
```

The following examples will show the `for..next` loop in operation.

7.2.2 Example 1: Converting inches to centimetres

Problem: Write a program that will convert inches to centimetres for a range of values and using an increment of 5. Use the `for..loop` and output the results in tabular form. Use the conversion factor 2.54 centimetres to the inch.

Solution

The algorithm is shown below:

```
1 display table titles
2 for each value from 1 to 50 step 5
3 calculate conversion to centimetres using the factor 2.54
4 tabulate output
5 next counter
```

The full Visual BASIC program is seen in Code 20.

```
Option Explicit

Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate the For..Next loop

  Dim Inches As Integer
  Dim Cms As Single

  PicDisplay.Print Tab(3); "Inches"; Tab(12); "Centimetres"
  PicDisplay.Print

    For Inches = 0 To 50 Step 5
       Cms = Inches * 2.54
       PicDisplay.Print Tab(5); Inches; Tab(15); Cms
    Next Inches

End Sub
```

Code 20

The program output is shown in Figure 7.2

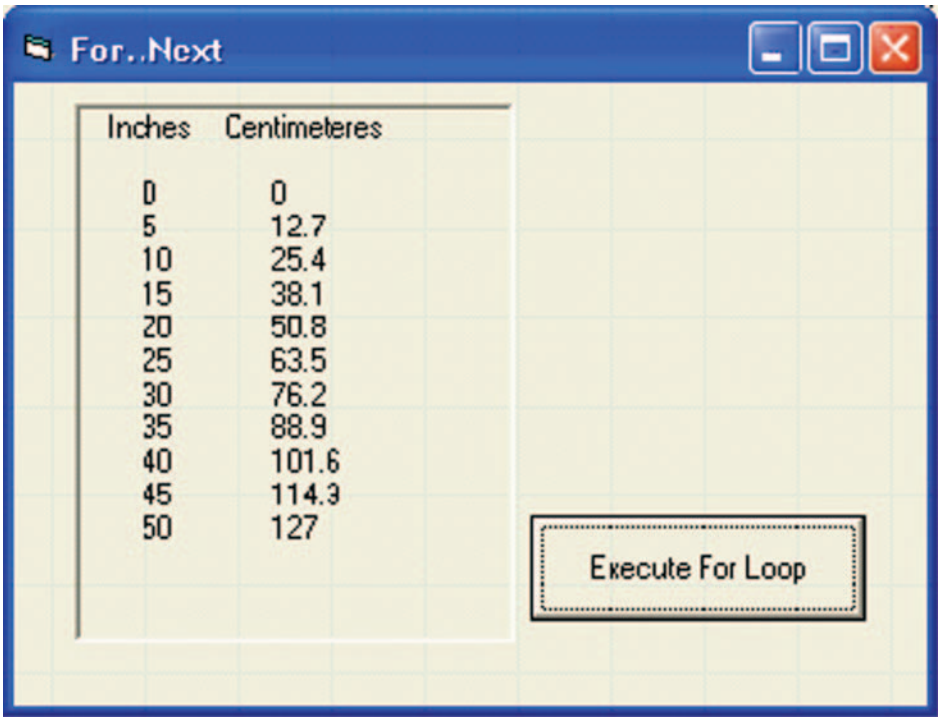


Figure 7.2:

Note that the output of this program is in tabular form. This was achieved using the `TAB()` function, first to output the heading then the results. Tab is short for tabulate and

the output items are separated by the values within the tab statements.

If the value of `step` is not specified it is assumed by Visual BASIC to be the value 1 by default.

Conversion program

1. Code and test the conversion program.
2. Adapt it to convert miles to kilometres. (use 1 mile = 1.609 km)

7.2.3 Example 2: Displaying integers using negative step

Problem: Write a program that will output ASCII codes from 100 to 65 together with the corresponding characters that the numbers represent. Use a for..next loop with -ve step and output the results in tabular form.

Solution:

```
1. display "The ASCII codes from 100 to 65"
2. for counter = 100 to 65 step -1
3. display the value of the integer and character
4. next counter
```

The Visual BASIC program is shown in Code 21.

```
Option Explicit

Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate the For..Next loop using -ve step

  Dim Counter As Integer
  Dim Char As String

  PicDisplay.Print Spc(3); "ASCII codes"; Spc(12); "Character"
  PicDisplay.Print

    For Counter = 90 To 65 Step -1
      Char = Chr$(Counter)
      PicDisplay.Print Spc(5); Counter; Spc(15); Char
    Next Counter

End Sub
```

Code 21

Note that, in this case the output has been formatted using the `SPC()` function. This allocates a number of spaces between output items depending on the value expressed within the function.

The program also uses the `CHR$()` function to convert a numerical value to its corresponding character.

For example: `Chr$(65)` returns the character "A"

Part of the program output is shown in Figure 7.3

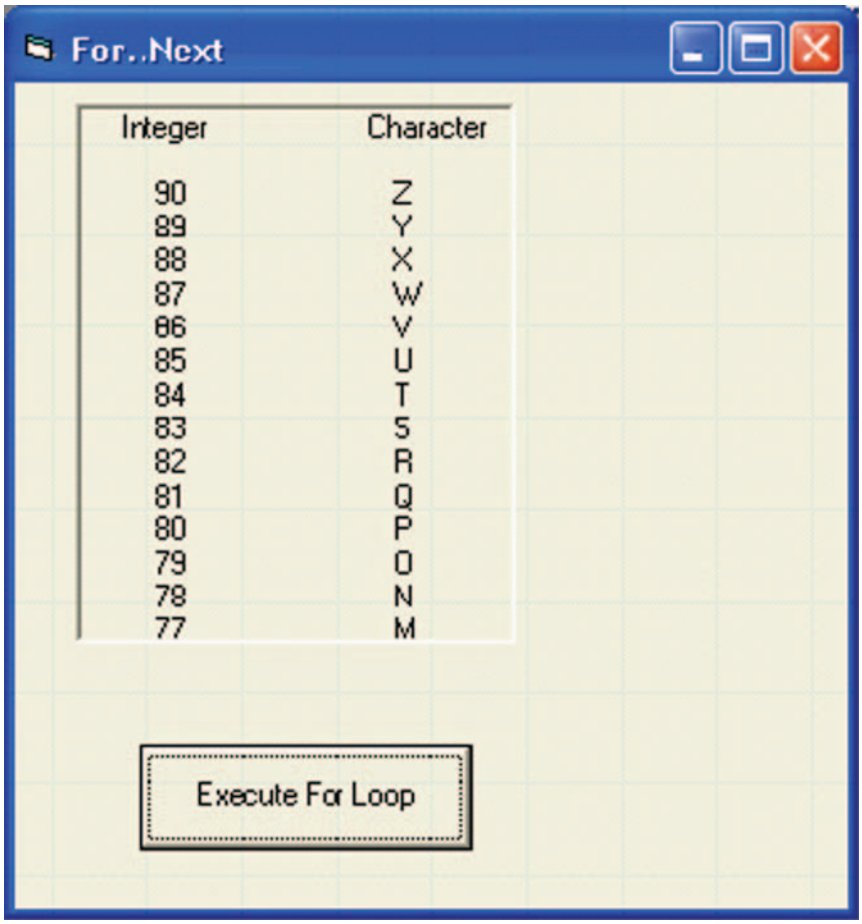


Figure 7.3:

ASCII code program

1. Code the program above.
2. Adapt the program to display any range of ASCII codes and characters, using Input Boxes to get the first and last number form the user.

7.3 Nested For loops

`For..Next` loops can be nested to allow the programming of loops within loops.

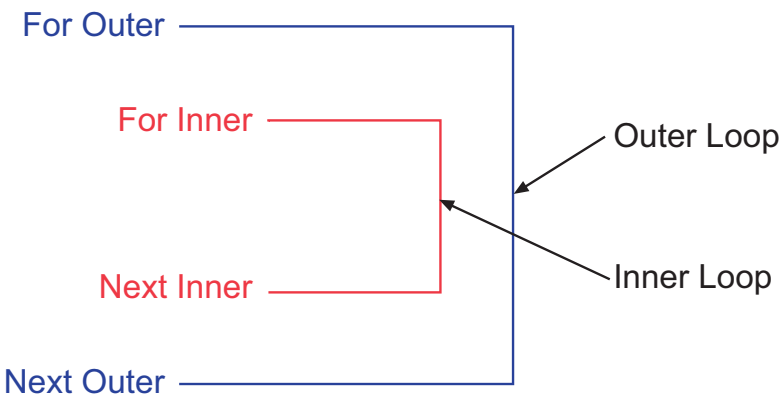
Consider the Visual BASIC program in Code 22. See if you can visualise what the output will be before looking at the results screen:

```
Option Explicit

Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate nested loops

  Dim Outer As Integer, Inner As Integer

    For Outer = 1 To 15
       For Inner = Outer To 15
          PicDisplay.Print Inner;
       Next Inner
       PicDisplay.Print
    Next Outer

End Sub
```

This file (NestedIF2.txt), can be downloaded from the course web site.

Code 22

In this program there are two loops that are controlled by the variables `Outer` and `Inner`.



The outer loop is initialised with the variable `Outer = 1`. The inner, nested loop is now executed 15 times and the first line of numbers are printed on the same line. This is achieved by using the semi colon at the end of the first print statement. `Outer` then takes on the `value 2` and the process repeats itself until `Outer = 15`.

Each time the output is decreased by 1 as the outer loop is executed until the value 15 is reached.

The print statement on its own ensures that a new line is taken for the next row of output. The print statement on its own basically means a line feed.

The output of the program is shown in Figure 7.4:

Note that it is considered bad programming practice to jump out of loops without terminating them fully. After the loops terminate the counter variables are discarded so if this is aborted prematurely, program output may not be as expected.

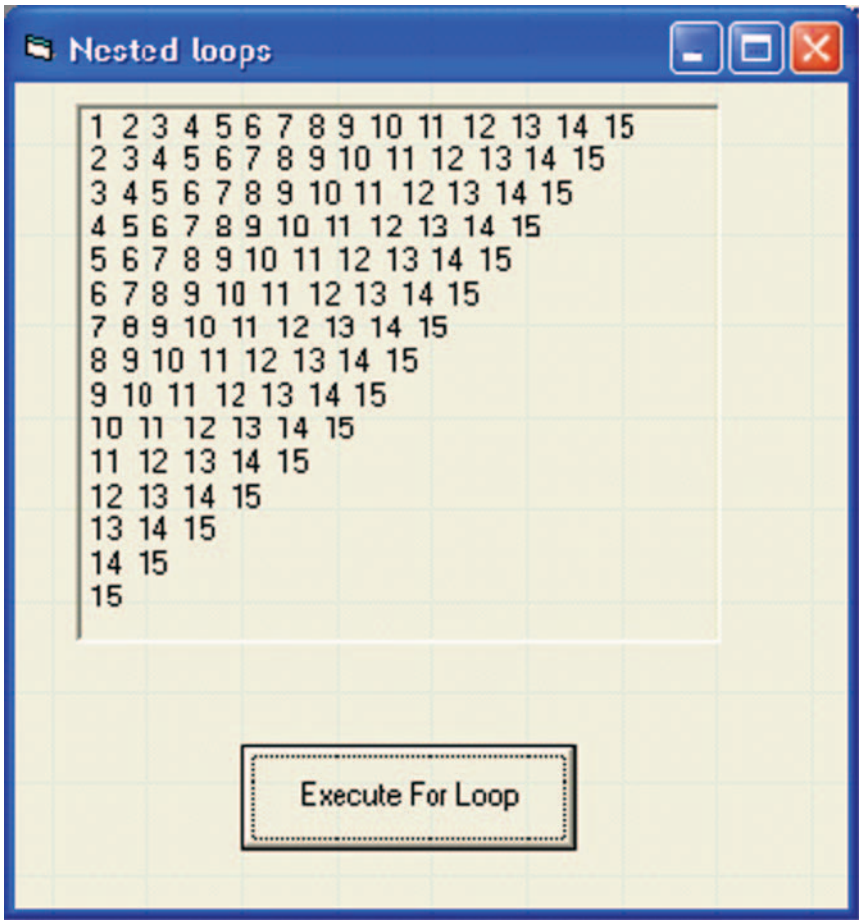


Figure 7.4:

Nested loop program

Experiment with the loops and range of numbers to arrive at different outputs.

7.3.1 Use of an If statement with nested For..Next loop

Problem: Write a program that calculates bank interest on a sum of money that is input by the user. Output the capital sum together with the interest.

Solution:

Consider the following algorithm:

```
1 Ask user to input value
2 If value <> 0
3  for count = 1 to 10
4   calculate interest on value
5   output value and interest
6  next count
7  else output message "Value not valid - try again!"
8 end if
```

The Visual BASIC Code 23 is shown:

```
Option Explicit
Private Sub Command1_Click()
  '16th February 2004
  'Program by Fred Fink

  'This program will illustrate nested loop and IF

  Dim Capital As Integer, Counter As Integer
  Dim Interest As Single

  Capital = InputBox("Please input capital amount")
  SumDisplay.Print Capital
  If Capital <> 0 Then
    PicDisplay.Print
    For Counter = 1 To 10
      Interest = (Capital * Counter) / 100
      PicDisplay.Print Tab(5); "$" & Capital; Tab(12); "  will  gain
         " & "$" & Interest & "   at " & Counter & "%"

    Next Counter
    Else
    PicDisplay.Print "$" & Capital & " is not a valid input. Please try again!"
  End If
End Sub
```

Code 23

The program involves an `If` statement with an embedded `for..next` statement. If the capital sum entered does not meet the initial condition then control will be passed to the `else statement`. If the condition is met then the program will continue and execute the `for..next` statement and output results.

Note the output line which looks rather complex. All it is doing is concatenating the output as a mixture of string and program variables. Concatenation is not just reserved for strings.

Bank Interest Program

Code and run the program.
You should end up with output as shown in Figure 7.5 for N = 10.

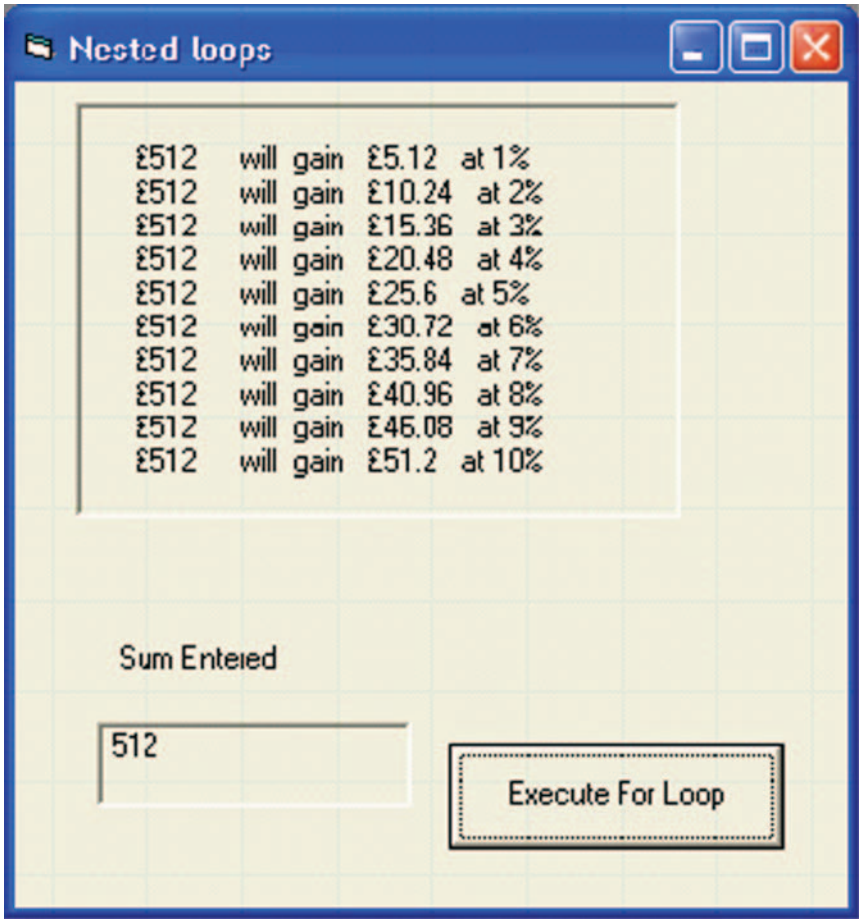


Figure 7.5:

7.4 Fixed Loop Exercises

There are three exercises here:

- Mowing meadows,
- Drawing right-angled triangles and,
- the hardest, Pythagoras' Theorem

Mowing meadows


30 min

Use a nested loop to generate the following lines of a well known verse:

```
1 man went to mow a meadow
2 men, 1 man went to mow a meadow
3 men, 2 men, 1 man went to mow a meadow
...
N men, N-1 men, N-2 men, ....., 1 man went to mow a meadow
```

Drawing right-angled triangles


30 min

Use a nested loop to draw right-angled triangle as shown below:

```
*
***
*****
*******
*********
```

Pythagoras' Theorem


40 min

Write a program that will output integer values representing the sides of right-angled triangles that satisfy Pythagoras's theorem i.e

```
a^2  +  b^2   =   c^2
```

Only output values that satisfy the equation.

Hint: You will require three nested `for..next` loops and experiment with loop values up to 10 for each loop otherwise the program may run out of memory.

7.5 Review Questions

Q4: Which one of the following describes correctly an incremental **for** loop?

a) Control variable is decreasing in value by a variable amount
b) Control loop variable is increasing in value by 1
c) Control loop variable is increasing by a variable amount
d) Control loop variable is increasing in value by a constant amount determined by the programmer

Q5: Which one of the following statements is not permitted?

a) For loopCounter = (3*4) to (5*6) step 1
b) For loopCounter = (3*4) to (5*1) step -1
c) For loopCounter = (3*4) to (5*2) step 1
d) For loopCounter = (3*4.16) to (5*5.6) step 1

Q6: Which one of the following problems is best suited to the use of a FOR loop?

a) Calculating the total number of marks entered at a keyboard
b) As an event loop that checks for keyboard input
c) Calculating the average of marks held in a file
d) All of the above

Q7: What shape will be displayed by the following Visual BASIC program fragment for any value of N $>$ 1?

```
for i = 1 to N
    for j = 1 to i
        print "+";
    next j
    print
next i
```

a) Triangle
b) Square
c) Rectangle
d) Circle

Q8: What will be displayed by the following Visual BASIC program fragment, assuming N = 3

```
sum = 0
for i = 1 to N
    for j = 1 to N
        sum = sum + j
    next j
next i
print sum
```

a) 16
b) 17
c) 18
d) 19

7.6 Conditional Loops

With the `for..next` loop the number of iterations must be known in advance so that the counter variable can be set.

There are many occasions in programming where the number of iterations is unknown so an alternative looping structure has to be used. The `Do...Loop` is a viable alternative to the `for..next` statement.

In Visual BASIC, `Do..loops` come in a variety of flavours and for any given program there will probably be more than one solution using a `Do..loop` variant.

There are two main `do..loop` constructs with two variants::

1. `Do While...loop` (and `variant Do loop..While`)
2. `Do Until...loop` (and `variant Do loop..Until`)

7.6.1 Do While..loop

The `do while` loop repeats a given set of instructions *while* a given condition is true.

The general form of this statement is:

```
Do while test condition
  statements
Loop
```

The loop repeats itself until the condition becomes `false`.

If the condition is false to begin with then the `do..while` loop will not be entered and control will pass to the rest of the program.

See Figure 7.6

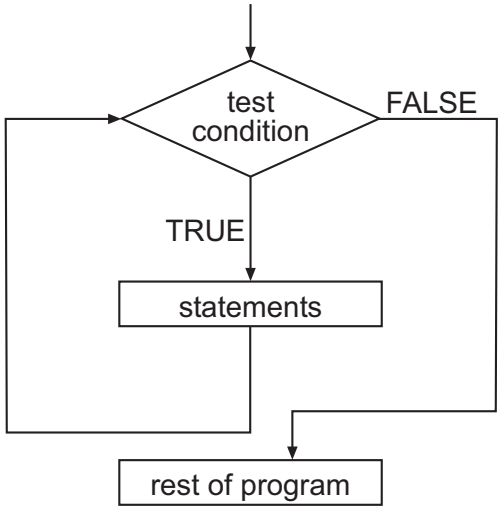


Figure 7.6:

The `do while..loop` is an example of a top tested *while true* structure. The variant do loop..while is an example of a bottom tested *while true* structure:

```
Do
  statements
Loop while test condition
```

The top tested loop can iterate between 0 and N times whereas the bottom tested loop may iterate between 1 and N. This means that using the bottom tested loop the iteration must occur at least once, so the condition can be tested immediately.

Care should be taken when writing programs using loops as an incorrect condition, or mistakes in the body of the loop can cause the program to get stuck in an infinite loop when executing, even when compilation produced no errors. Should this occur in Visual BASIC simply click on program `Run` and choose `End`. In more extreme cases pressing `ctrl + alt + delete` will allow you to abort Visual BASIC.

7.6.2 Example 1 - Calculating a sum of positive integers

Problem: A program is required to accept positive numbers from the keyboard, calculate and display a sum of all the numbers entered. It should use a do while..loop which tests whether each number entered is greater than or equal to zero. If the user enters a negative number then the loop will terminate and the total will be displayed on the screen.

Solution:

The algorithm is shown below:

```
1. set the running total to 0
2. display "Give me your first number"
3. get number typed in at the keyboard
4. do while user's number >= 0
5. add number to the running total
6. display "Give me the next number"
7. get input typed in at the keyboard
8. loop
9. display the total
```

The Visual BASIC program is shown as Code 24.

```
Option Explicit

Private Sub Command1_Click()
  '17th February 2004
  'Program by Fred Fink

  'This program will illustrate the Do While..loop

  'program addPos

  'This program will add a list of positive numbers typed in
  'at the keyboard. The program will stop when a negative number
  'is entered and display the total
```

```
Dim iNumber As Integer, Total As Integer

  Total = 0
  iNumber = InputBox("Give me your first number ")
  Do While iNumber >= 0
      Total = Total + iNumber
      iNumber = InputBox("Give me your Next number")
      PicDisplay.Print ("Your numbers add up to "); Total
  Loop

End Sub
```

This file (DoWhile.txt), can be downloaded from the course web site.

Code 24

The program output is shown in Figure 7.7

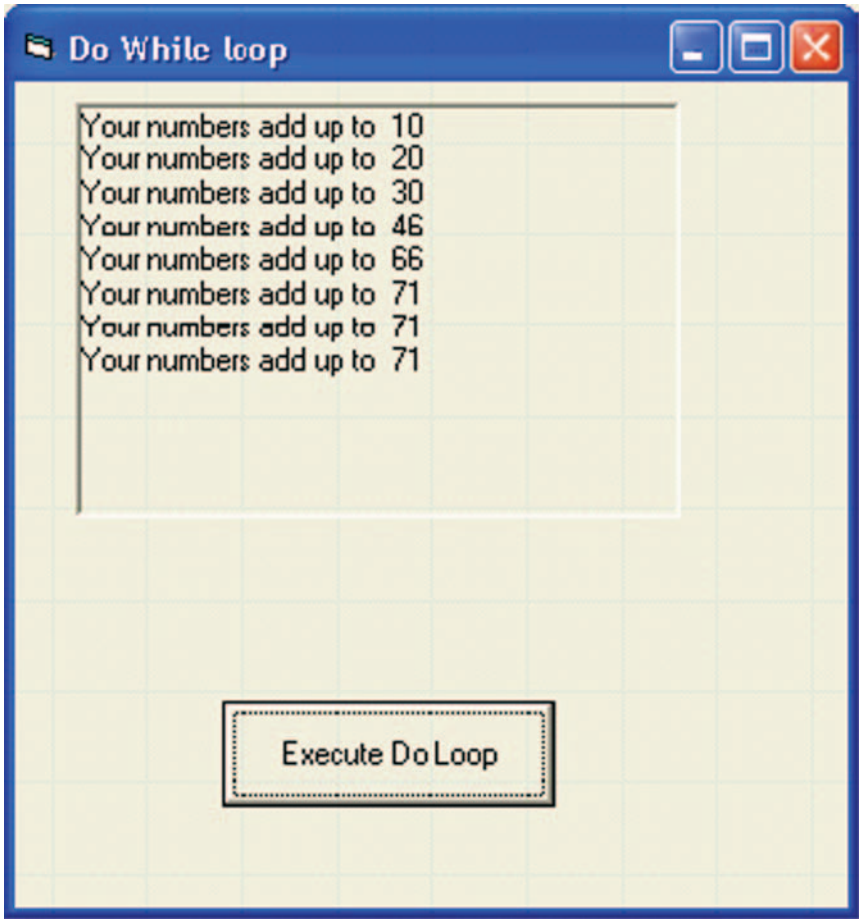


Figure 7.7:

Running total program

Code, run and test the progam above.

7.6.3 Example 2 - Validating character input problem

A program is written to prompt the user to enter a character. The program continues to prompt until it receives a character other than a 'Y' or a 'y'.

Solution:

The algorithm is shown below:

```
1. display "Continue ?"
2. get input from the keyboard
3. do while user enters "Y" or user enters "y"
4. display "Continue"
5. get input from the keyboard
6. loop
7. display "Out of loop"
```

The Visual BASIC program is shown in Code 25.

```
Option Explicit

Private Sub Command1_Click()
  '17th February 2004
  'Program by Fred Fink

  'This program will illustrate the Do While..loop

  'Program CharEntry

  'This program prompts the user to enter a character.
  'It uses a WHILE loop to repeatedly check that a
  'character other than a 'Y' or a 'y' has been entered

  Dim Response As String

  PicDisplay.Print ("Do you want to continue ?")
  Response = InputBox("Y or y")
  Do While (Response = "Y") Or (Response = "y")
    Response = InputBox("Continue ?")
  Loop
  PicDisplay.Print ("Out of loop")

End Sub
```

This file (Validate.txt), can be downloaded from the course web site.

Code 25

The program output is minimal; since it only responds to the correct input then the program will simply accept the character that is entered. Only when the incorrect character is entered will the program fail to enter the loop and come up with the message "Out of loop".

This code can be used as part of a larger program to validate input. If coded as a procedure then it can be called from within the main program. We will discuss procedures more fully later in the final topic.

Input validation program

Code, run and test the program above.

7.6.4 Example 3 - Range checking using a boolean variable

Problem: A program is required to ask the user to enter a value. If the value is outside the range expected the user will be prompted to enter another value. (This is another example of validation, only in this program the use of a boolean variable is exemplified.)

Solution:

The algorithm is shown below:

```
1. define upper and lower limits
2. set value of ok to false
3. Ask user to input a number
4. do while ok = false
5.   if (number >= low value) and (number <= high value) then
6.     set ok to true
7.     display "This number is in range"
8.   else
9.     display "This number is out of range"
10.    display "Try entering another number"
11.    input "What number is needed?"
12.  end if
13. loop
14. display "Out of loop"
```

The Visual BASIC program is shown in Code 26.

```
Option Explicit

Private Sub Command1_Click()
  'program RangeCheck
  '18th February 2004
  'Program by Fred Fink

  'This program prompts the user to enter numeric value.
  'It uses a WHILE loop to repeatedly check that a
  'value which is between LowValue and HighValue

  Const LowValue As Integer = 10
  Const HighValue As Integer = 20

  Dim size As Integer
  Dim OK As Boolean
```

```
  OK = False
  size = InputBox("Enter a number")
  Do While Not OK
    If (number >= LowValue) And (number <= HighValue) Then
      OK = True
      Print number; " is in range"
       Else
      Print number; " is out of range, try again"
      size = InputBox("What number is required?")
    End If
  Loop
Print ("Out of loop")
End Sub
```

This file (RangeCheck.txt), can be downloaded from the course web site.

Code 26

Sample program output is seen in Figure 7.8

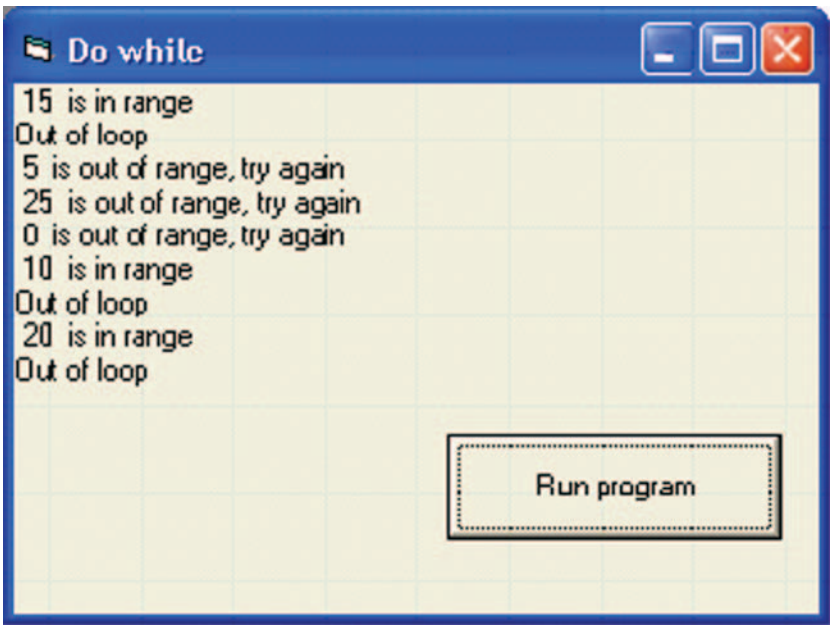


Figure 7.8:

Range check program

Code, run and test the program above.

7.6.5 Do..While Review Questions

Q9: What is wrong with the following?

```
n = 0  'assume n is an integer
do while n < 100
    value = n * n
loop
```

Q10: What is wrong with the following?

```
i = 1  'assume i is an integer
do while i <= 10
    Print(i)
i = i + 1
```

Q11: Write a `while` loop that calculates the sum of all numbers between 0 and 20 inclusive. Hint: use two integer variables, one for a loop counter and one for keeping a running total of the numbers

7.6.6 Do Until.. Loop

The `Do While` loop performs the **conditional test first** and then executes the loop, so the statements within a loop may never be executed.

The `do until` loop performs the **statements first** and then tests the condition. This means that the body of the loop is always executed at least once.

This is the only difference, but a significant one between these looping constructs.

The general form of a `do until.. loop` is:

```
Do Until test condition
  statements
loop
```

The statements in the body of the loop are executed repeatedly until the test condition is FALSE.

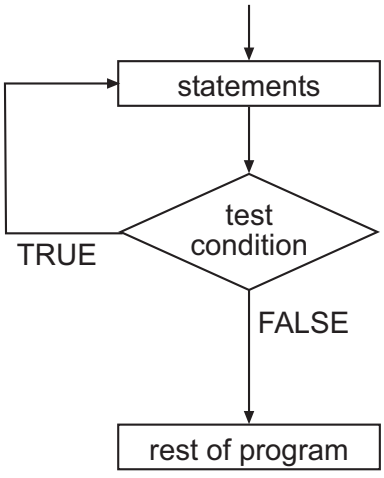


Figure 7.9:

The `do until.. loop` is an example of a top tested *while false* structure.

The variant `do loop..until` is an example of a bottom tested *while false* structure:

```
Do
   statements
Loop until test condition
```

The top tested loop can iterate between 0 and N times whereas the bottom tested loop may iterate between 1 and N. This means that using the bottom tested loop the iteration must occur at least once, so the condition can be tested immediately.

Some simple examples should clarify the situation.

Consider the simple lines of code representing a simple `do until..loop`

```
Dim Num As Integer

Num = 1
Do Until Num = 10
  Num = Num + 1
Loop

MsgBox Num
```

The equivalent `do while..loop` would be:

```
Dim Num As Integer

Num = 1
Do While Num < 10
  Num = Num + 1
Loop

MsgBox Num
```

Although both loops end up with similar results their methods are different. With the `do until loop` the condition `Num = 10` is set and the loop must be entered at least once to test this condition as `true`. `Num` is continually tested until the condition fails i.e. `Num` becomes $>$ 10 and the loop exits with `Num = 10`.

The equivalent `do..while` loop tests the condition in the first line and if `false` the loop will not be entered. If true then the loop will run until `Num = 10` when the loop exits.

When should you use one and when the other?

If you know it is safe to run the code at least once, probably you should use the `do..until` loop. If you *must* run the code at least once then again a `do..until` loop is a good solution.

If there is any reason to doubt the value of any variables etc. in the loop, then you should

always use the `do..while` loop. Menus often use `do..until` loops as you know that the menu needs to be run at least once for the user to see it!

A number of worked solutions to problems are given here - make sure you understand what is going on in these examples. You may wish to try running these programs for yourself and to experiment with making changes to the code to ensure you fully understand what is happening.

7.6.7 Example 1: Guessing an age with `do..until` and nested `else..If`

Problem: A program is required to ask the user to guess an age, the value of which is stored as a program constant. The program should count the number of tries needed to guess the correct age and declares whether the guesses are high or low.

Solution:

The algorithm is shown below:

```
1. set the number of guesses to 0
2. do
3.   add 1 to the number of guesses
4.   request an age from the user
5.   if guess > age output message "too high"
6.   elsif guess < age output message "too low"
7.   endif
8. loop until guess = age
9. display "The number of times guessed"
```

The Visual BASIC program is shown in Code 27.

```
Option Explicit

Private Sub Command1_Click()

'program MyAge

'17th February 2004
'Program by Fred Fink

'This program will prompt the user to guess an age which is stored
'in the program as a constant.  It will display the number of
'attempts needed to guess the correct age

Const MyAge As Integer = 21 'again

Dim ThisGuess As Integer, guesses As Integer

  guesses = 0
  Do
    ThisGuess = InputBox("Guess my age")
    guesses = guesses + 1
```

```
    If ThisGuess > MyAge Then
       PicDisplay.Print ThisGuess; " is too high! "
    ElseIf ThisGuess < MyAge Then
       PicDisplay.Print ThisGuess; " is too low! "
    End If
  Loop Until ThisGuess = MyAge
    PicDisplay.Print "Ok,so its "; MyAge & " but it took you ",
guesses; " tries"
End Sub
```

This file (GuessAge.txt), can be downloaded from the course web site.

Code 27

The output is shown in Figure 7.10

Figure 7.10:

Guess my Age program

Code, run and test the program above.

7.6.8 Example 2: Fibonacci numbers

Problem: Write a program to output numbers belonging to the Fibonacci series up to a specified maximum.

Fibonacci was a famous Italian mathematician who identified the following series of numbers:

```
1, 1, 2, 3, 5, 8, 13.......
```

Successive terms of the series are calculated by adding the previous two numbers.

Solution

The following algorithm will produce the series, given the first two values as input.

```
1.  Do
2.    input number of terms to output
3.   loop until input is within range
4.  input two starting values
5.  do
5.    compute numbers
7.    display numbers
8.  loop until number of terms have been output
```

```
Option Explicit
Private Sub Command1_Click()
  'program Fibonacci

  '17th February 2004
  'Program by Fred Fink

  'This program will prompt the user to input the maximum value
  'of terms to be displayed. Two starting values
  'are also input.

  Dim NoOfTerms As Integer, First As Integer, Second As Integer
  Dim Third As Integer, Count As Integer
  PicDisplay.Cls
  PicDisplay2.Cls
  Do
    NoOfTerms = InputBox("How many terms to display?") 'Validation check
  Loop Until (NoOfTerms > 0) And (NoOfTerms <= 16)
  PicDisplay2.Print NoOfTerms
  First = InputBox("Enter first number")
  Second = InputBox("Enter second number")
  Third = 0
  Count = 0
  PicDisplay.Print First; Second;
  Do
    Third = First + Second
    First = Third
    Second = Third + Second
    PicDisplay.Print First; Second;
    Count = Count + 1
  Loop Until Count = (NoOfTerms - 2) \ 2
End Sub
```

This file (Fibonacci.txt), can be downloaded from the course web site.

Code 28

The program input is restricted to values $>$ 0 and $<$ 17 using a DO .. Until loop for input validation.

If you find difficulty in following the logic of the program then perform a paper exercise running through the values of each variable as the program executes. This is called a *dry run* and is best done by means of a *trace table*:

Instructions	First	Second	Third	Count	Output
Starting Values	1	1	0	0	1, 1
First Loop	2	3	2	1	2, 3
Second Loop	5	8	5	2	5, 8
Third Loop	13	21	13	3	13, 21

If you have access to the on-line version of SCHOLAR, you will find it helpful to view the interactive version of this table, which shows how it is built up line by line during a dry run of the code.

The two starting values are 1, 1 which are assigned to variables `First` and `Second`. Variable `Third` then takes on the value of `First + Second` which is 2. First now takes on the value of `Third` to become 2. Finally the variable `Second` takes on the value of `Third + Second` to become 3. The values of `First` and `Second` are now displayed to give the output:

```
1,  1,  2,  3,  5,  8........
```

Because the output is `First` and `Second` the value of `Count` is halved otherwise the output would be twice that required i.e.

```
Do Until Count = (NoOfTerms - 2) \ 2
```

Also the value of `NoOfTerms` is decreased by 2 to take into account the first two values which are output first and are not part of the loop.

Sample program output is shown in Figure 7.11

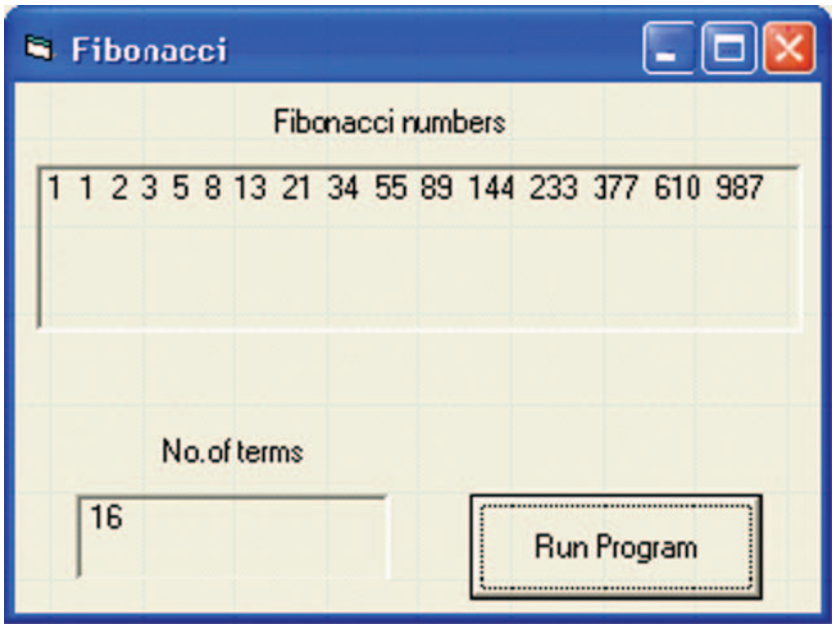


Figure 7.11:

7.6.9 Review Questions

Q12: Which one of the following statements is true regarding the **do while** loop?

a) The conditional statement is always satisfied
b) The block of code is entered first before the condition is tested
c) The loop need not be entered if the condition fails at the start
d) The loop terminates when the condition becomes true

Q13: Looping structures have several features in common. Which one of the following is NOT one of these features?

a) Selection
b) Increment
c) Initialisation
d) Condition

Q14: What is the output of the following program fragment that uses a **do while** loop?

```
i = 0
Do
    i = i + 1
    Print i;
Loop While i < 5
```

a) 0 1 2 3 4
b) 1 2 3 4 5
c) 0 1 2 3 4 5
d) 1 2 3 4 5 6

Q15: In the previous example the **do while** loop is replaced by a **do until** loop as follows:

```
i = 0
Do
    i = i + 1
    Print i;
Loop Until i < 5
```

a) 0
b) 1
c) 0 1 2 3 4 5
d) 1 2 3 4 5

Q16: What change can be made to the program fragment in the previous question to give the output 1 2 3 4 5?

a) Change last statement to Loop Until NOT i $<$ 5
b) Change last statement to Loop Until i $>$= 5
c) Use a for loop with counter variable i
d) Any one of the above

7.7 Formatting output

Up to this point it has been left to Visual BASIC to output data in default mode. However it is possible to have more control over how `real values, currency, boolean` variables are output by using the Visual BASIC in-built function `Format()`.

The structure of the Format function is:

```
Format (variable, format expression)
```

Table 7.1 shows the formatting functions within Visual BASIC:

Table 7.1:

Format name	**Meaning**	**Examples**
General	Displays raw number without separators	12345
Fixed	Displays at least one digit before the decimal point and two digits after the point	67.88
Scientific	Uses scientific notation	6.023 E23
Standard	Displays numbers with separators and two digits after the decimal point	1,234.56
Currency	Same as standard. Negative values are enclosed within parentheses (brackets)	(1,234.56)
Percent	Displays numbers multiplied by 100 with two digits after the decimal point followed by the % sign	12345.67%

Exercise - Output formats

Write a Visual BASIC program to input real values and output the value in each of the formats in Table 7.1. Use the algorithm below to help you. Make the value large enough so that all aspects of the formatting can be shown.

The program should terminate when 0 is entered.

```
1 Do
2   input a real number
3   display in general format
4   display in fixed format
5   display in scientific format
6   display in standard format
7   display in currency format
8   display in percent format
9 Until number = 0
```

Run the program a few times with different values so that you understand the nature of each format.

Custom formats

The Print Format allows you to create your own formats, using various format characters:

0 means display a zero or a digit
means display a digit or nothing
% means multiply by 100 and insert a percentage character

You can also add in any of the following characters: . , - + $ ( )

For example: Print "Using '##,##0.00':"; Tab(25); Format$(Number, "##,##0.00")

See www.vb6.us/tutorials/understanding-vb-format-function-custom-numeric-formats for more examples.

Adapt your program to experiment with this function.

7.8 Arrays

In the exercises so far you have looked at simple data types, such as `single(real)` and `integer`. The next thing we want to look at is how related data items can be stored together using *arrays*.

Much of this material may be familiar to you already. However, it is necessary to gain more practice in the use of arrays to prepare you for a later topic on *standard algorithms*, which makes extensive use of this type of data structure.

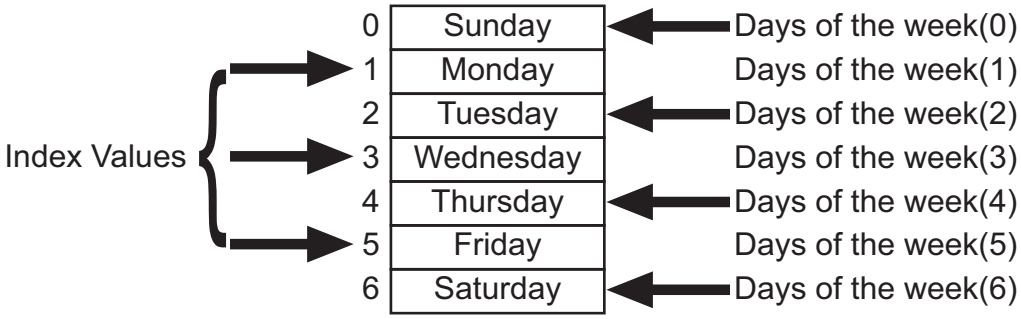
A data structure is a way of storing data in a computer in an organised way. One of the simplest data structures used in computer programming is called an array and is an example of a static data structure because it is of a fixed size within memory as defined within the structure of the program.

An array is a list of data items where each item is uniquely identified by its position in the list. An array is given a name, which is usually related to the group of data it holds. Some examples might be:

Data Items	**Array Name**
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday	'Days of the week'
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	'Hexadecimal digits'
£1, £2, 50p, 20p, 10p, 5p, 2p, 1p	'Decimal coinage'

7.8.1 1-D Arrays

The data items in 1-D arrays are simply stored in consecutive locations within a block of computer memory as follows:



Thus the data item '**Tuesday**' is stored in array location **Days_of_the_week(2)** and the data item '**Saturday**' as `Days_of_the_week(6)`. Each data item therefore is uniquely identified by its index.

Note that the index values are not actual memory addresses as such but simply refer to the data positions within the array. In Visual BASIC, the first element usually has index 0.

Imagine you were to process the average temperature data in Edinburgh for the last 30 days. If each of the day's temperatures are stored as separate variables you would require 30 distinct variables, and the storage and manipulation of these variables becomes difficult.

To overcome the above problem, you can define a variable called `temperature` which represents not a single value of temperature but an entire *set of temperatures*.

This is illustrated in Figure 7.12 showing an array structure called temperature and elements with the subscript ranging from 0 to 8.

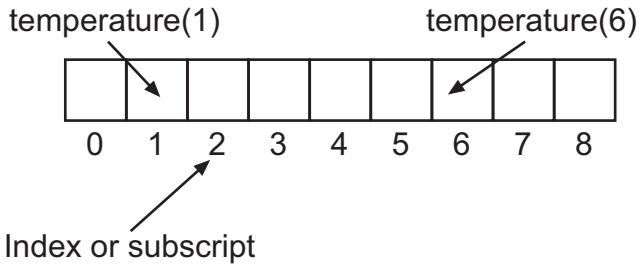


Figure 7.12: Representation of an array and indices

An individual array element can be used anywhere that a normal variable could be e.g.

```
temp = temperature(27)
average = (temperature(5) + temperature(6))/2.0
```

A value can be stored in an array simply by specifying the array element on the left hand side of the assignment operator (equals sign), e.g.

```
temperature(2) = 18.0
```

assigns the value 18.0 to be stored in element with subscript 2 of the `temperature` array.

```
temperature(7) = 14.5
```

assigns the value 14.5 to be stored in element index number 7 of the `temperature` array.

This is illustrated in Figure 7.13

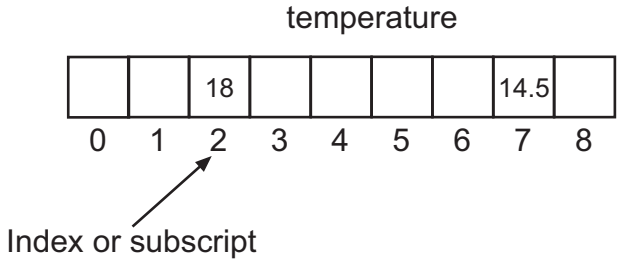


Figure 7.13: Representation of the temperature array with temperature(2) = 18.0 and temperature(7) = 14.5

More generally, an *integer* variable can be declared and used as the subscript, e.g.

```
temp = temperature(i)
```

which will take the value assigned in element number `i` of the array `temperature` and assign it to the variable `temp`.

This means that if you want to access the elements of an array sequentially, you can do it using a `for..next` loop where the variable for the subscript is automatically incremented each time you go round the loop. Remember that the subscript variable can only be an `integer`. A `real` is not allowed. The examples within this topic will show you how you can access an array using a loop. This is the most common way of marching through an array from beginning to end, and as you will probably use this technique a lot in your programs.

7.8.2 Declaring arrays

Before you can use an array it must be declared at the beginning of the program with the other variable declarations. The declaration provides:

- the name of the array,
- the number of elements and
- the data type of the elements.

Like variables, arrays can be declared using the Visual BASIC keywords `Dim`, `Public`, and `Private` that determines their scope.

If `Dim` is used then the array is private to the procedure in which it is declared.

`Public` makes the array visible from anywhere in the program.

`Private` (within the General section of a form or module) makes the array visible only to the form or module in which it's declared.

Note that in Visual BASIC, when an array is declared the first index is 0 by default. It's possible in some versions of Visual BASIC, however, to force the first element of an array to be 1 by using the To statement.

All this will make sense with a few examples:

1. Declare an array called `MyName` to hold 6 integer values

```
Dim MyArray(5) As Integer
```

2. Declare an array called `Names` to hold 3 string values

```
Dim Names(1 to 3) As String
```

3. Declare a public array called `Numbers` to hold 50 decimal values

```
Public Numbers(49) as single
```

Note: Visual BASIC.NET does not support declarations using "to".

Common errors associated with arrays

1. Note that you cannot mix the types of data stored in an array. So a single array cannot be used to store integer, real or string variables together.

2. The most common error is to attempt to access an array element outwith the declared bounds e.g. accessing element `0` or `15` of an array with subscript range specified as `1..14`.

This type of error is so common that it has been given the name 'fencepost error'. It is not a minor error as it can crash the system.

7.8.3 Simple 1-D array manipulation

Example 1 - Using an integer variable to initialise an array

Problem: How can the array temperature be initialised to zero using a for loop?

Solution: A typical solution to this problem is shown here.

```
for count = 0 to 4
  temperature(count) = 0
next count
```

The first time round the loop the variable `count` has the value 0 and so the element with index number 0 of the array `temperature` is set to 0. The next time round the loop `count` has been incremented by 1 and has the value 1 and so element index number 1 of the array will be set to 0 and so on.

This one simple loop replaces 5 lines of code, i.e.

```
temperature(0) = 0
temperature(1) = 0
temperature(2) = 0
temperature(3) = 0
temperature(4) = 0
```

Example 2 - Displaying the contents of an array

Problem: How can the contents of an array be displayed using a `for` loop?

Solution: If you assume you have an array called Store which has 6 elements then you could use the following code to display the contents of each element.

```
for count = 0 to 5
  Print("element "; count; "= "; Store(count))
next count
```

For example, if the array Store held the values (3, 12, -4.6, 3.2, 0, -1), see Figure 7.14:

Figure 7.14: Contents of a six-element array Store

Then the output would be:

```
Element 0 = 3
Element 1 = 12
Element 2 = -4.6
Element 3 = 3.2
Element 4 = 0
Element 5 = -1
```

This time one simple loop replaces 6 lines of code, i.e.

```
Print("element 0 = " Store(0))
Print("element 1 = " Store(1))
Print("element 2 = " Store(2))
Print("element 3 = " Store(3))
Print("element 4 = " Store(4))
Print("element 5 = " Store(5))
```

A single element can be displayed or accessed at random - you are not forced to process the entire array to do this! Any one of the six lines of code above does the job of printing out that particular array element, so the code `Print('element 5 = ', Score(5))` outputs the *sixth* element of the array - the first element being element 0.

Example 3 - Reading user input into an array

Problem: How can data entered by the user at the keyboard be stored directly into the elements of an array?

Solution: Consider the situation where you want to store 4 user entered integers in an array called `mark`. You could use a `for next` loop to prompt and obtain input and to store the integers in the array, e.g.

```
for times = 0 to 3
  mark(times) = InputBox("Please input marks")
next times
```

For each loop the user will be asked to input a mark. After four marks have been entered the loop will terminate and the array mark will be storing the four value.

30 min

Using arrays

Several examples using arrays have been illustrated in the course notes. Incorporate these program fragments into a working program which can do three things:

1. initialise each element of an array to zero;
2. allow you to input data in the form of floating point numbers directly into the array;
3. print out the contents of the array.

Try to make the program user-friendly by putting relevant information on the screen.

Finally, add a section to the above program which will print the array in reverse order after it has printed it conventionally.

20 min

Indexing arrays

Answer the following questions. Use the arrays interaction on the course web site to help you find out the answers.

Q17: Begin with an array of 4 elements with a subscript range of 0..3. All values are initialised to 0, i.e. array = [0, 0, 0, 0]. Set array[1] = 3. What are the arrays contents now?

a) array = [0, 0, 0, 0]
b) array = [3, 0, 0, 0]
c) array = [0, 3, 0, 0]
d) array = [0, 0, 3, 0]
e) array = [0, 0, 0, 3]
f) none of the above

Q18: Using the same array set array[0] = 7. What are the array contents now?

a) array = [0, 0, 0, 0]
b) array = [3, 7, 0, 0]
c) array = [3, 0, 7, 0]
d) array = [7, 3, 0, 0]
e) array = [0, 3, 7, 0]
f) array = [7, 0, 3, 0]
g) array = [0, 7, 3, 0]
h) array = [0, 7, 0, 3]
i) array = [0, 0, 7, 3]
j) none of the above

Q19: Set up a 4 element array with a subscript range of 0..3 with the following values: array = [3, 6, 2, 8]. What is the value of array[2]?

a) 3
b) 6
c) 2
d) 8
e) none of the above

Q20: In the same array as the previous question, what is the value of the element with index 0?

a) 3
b) 6
c) 2
d) 8
e) none of the above

Q21: Set up an 8 element array with a subscript range of 0..7, e.g.. array = [8, 23, 5, 19, 3, 0, 7, 52]. Now set array[1] = 12. What are the array contents now?

a) array = [8, 23, 5, 19, 3, 0, 7, 52]
b) array = [12, 23, 5, 19, 3, 0, 7, 52]
c) array = [8, 12, 5, 19, 3, 0, 7, 52]
d) array = [8, 23, 12, 19, 3, 0, 7, 52]
e) array = [8, 23, 5, 12, 3, 0, 7, 52]
f) array = [8, 23, 5, 19, 12, 0, 7, 52]
g) array = [8, 23, 5, 19, 3, 12, 7, 52]
h) array = [8, 23, 5, 19, 3, 0, 12, 52]

i) array = [8, 23, 5, 19, 3, 0, 7, 12]
j) none of the above

Q22: Again, starting with an 8 element array with a subscript range of 0..7 e.g. array = [8, 23, 5, 19, 3, 0, 7, 52]. Set array[3] = 9, element with index 6 = 11 and array[6] = 4. What are the array contents now?

a) array = [8, 23, 5, 9, 3, 0, 4, 52]
b) array = [8, 23, 5, 19, 9, 4, 7, 52]
c) array = [8, 23, 5, 9, 3, 4, 7, 52]
d) array = [8, 23, 5, 19, 9, 4, 4, 52]
e) none of the above

Q23: Set up an array that can hold 4 values with a subscript range of 0..3. Initialise the array to hold all 0 values. Now set the following values:

- array[1] = 9
- array[2] = 3

What are the contents of the array now? (express in format of array = [3, 2, 5, 0]).

Q24: Set up an array that can hold 10 values with a subscript range of 0..9 Initialise the array to hold all 0 values. Now set the following values:

- array[4] = 6
- array[8] = 2
- array[1] = 7
- array[7] = 3

What are the contents of the array now? (express in format of array = [3, 2, 5, 0]).

Q25: Write a program that will set each element in an array, called `myarray`, of 10 elements to the value of its index. The contents of the array are then printed out in a vertical line.

7.8.4 Review Questions

Q26: Initialising a 1-dimensional array means:

a) Setting all locations of an integer array to 0 (zero)
b) Setting all locations of a string array to null
c) Setting all locations of a real array to pre-determined values
d) All of the above options

Q27: The contents of a string array `message()` contain the following characters in successive memory locations:

H A P P Y B I R T H D A Y

The programming structure required to produce the above message could best be achieved using: (choose one)

a) A case statement
b) A for loop
c) A while loop
d) An until loop

Q28: An array `numbers()` is declared with 6 elements and initialised to contain the following values:

0 3 0 2 5 9

During a program run the array elements are changed as follows:

```
numbers(2) = 6
numbers(5) = 7
numbers(1) = 4
```

The array contents are now:

a) 0 3 0 2 5 9
b) 4 3 0 2 5 9
c) 4 6 0 2 7 0
d) 0 4 6 2 5 7

Q29: An array `value_1( )` has been declared in Visual BASIC to be of type single and to hold 5 values. Which one of the following statements would produce an error when the program is run?

a) value_1(0) = "Hello"
b) value_1(4) = 89.45
c) value_1(1) = 5.6E37
d) value_1(2) = 16

Q30: What is the least **positive** value of the variable `index` that will cause the following code to fail?

```
Dim array_1(19) As integer
Dim X As integer, index As integer
...
array_1(index) = array_1(index + index)
...
```

a) 9
b) 10
c) 11
d) 12

7.8.5 Calculating the average of the values stored in an array

Problem: Write a program to access each of the temperatures of a room over 14 days. Calculate the average temperature during the period.

Solution: First try: A typical solution to this problem (which doesn't use an array) is shown in Code 29.

```
Option Explicit
Private Sub Command1_Click()
  'program CalculateAverageTemp

  'program to calculate average temperature
  '20th February 2004
  'Program by Fred Fink

  Dim total As Single, average As Single, Temp As Single
  Dim day As Integer

  total = 0.0   'initialise total

  For day = 0 To 13
    Temp = InputBox("Enter temperature of day ")
    PicValues.Print day
    total = total + Temp
    PicArray.Print Temp
  Next day
  average = total / 14
  PicResult.Print Format(average, "standard")

End Sub
```

This file (AverageTemp.txt), can be downloaded from the course web site.

Code 29

Refinement - using an array

The program in code 7.10 gives correct output and result, but does not allow the programmer to re-use the data which has been entered. To do this, we need to use an array.

An improved version, using a 1-D array to store the data, is shown below:

```
Option Explicit
Private Sub Command1_Click()
  'program CalculateAverageTemp

  'program to calculate average temperature
  '20th February 2004
  'Program by Fred Fink

  Dim Store(13) As Single
  Dim total As Single, average As Single
  Dim day As Integer

  total = 0.0   'initialise total
```

```
    For day = 0 To 13
      Store(day) = InputBox("Enter temperature for the day")
      PicValues.Print day
      total = total + Store(day)
      PicArray.Print Store(day)
    Next day
    average = total / 14
    PicResult.Print Format(average, "standard")

End Sub
```

This file (AverageTemp2.txt), can be downloaded from the course web site.

Code 30

As the temperature values are entered they are stored in the array `Store()`. When the loop terminates the value of `average` is displayed.

The program output is shown in Figure 7.15

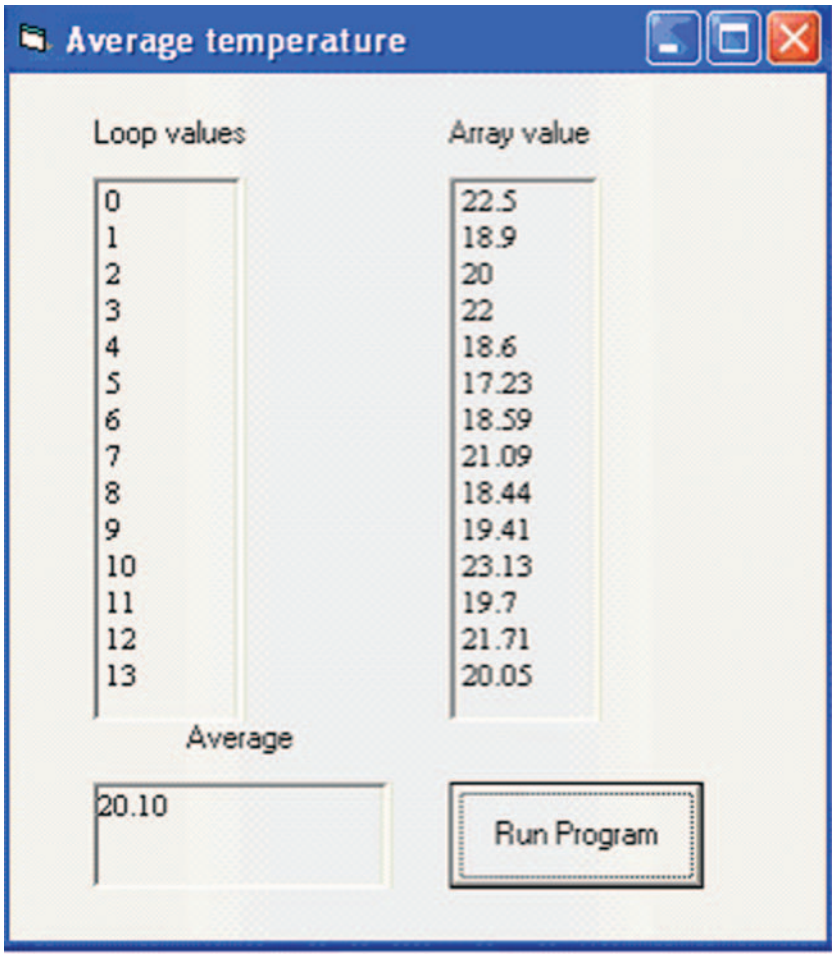


Figure 7.15:

Version 2

Adapt version 1 as above. Test the new version.

7.8.6 Random Events

Using arrays makes it easy to **simulate** many numerical activities that otherwise would be fairly tedious to do manually. For example analysing the results of tossing a coin 1000 times, throwing die or spinning a roulette wheel can all be dealt with in one-dimensional arrays. Such simulations make use of random numbers that can be generated between upper and lower limits for example:

tossing a coin: random values 1 (for heads) or 2 (for tails)

throwing a die: random values between 1 and 6

roulette: random values between 0 and 49.

Random numbers

Visual BASIC has a built-in function `RND()` that produces a random number from 0 to 1. The `randomise` function allows the `RND` function to start from a seed value and to produce a series of numbers based on the seed. Random numbers are generated internally using the computer's internal clock.

Since the random values produced are real, they must be converted into integers before being stored in the array. This can be achieved using another function `INT()`. Values are rounded down to the nearest integer.

For example:

```
Int(7.3)   =  7
Int(-6.8)  = -7
```

7.8.7 Simulation of tossing a coin

Problem: Write a program to simulate the tossing of a coin up to1000 times determined by the user and output the number of heads and tails produced.

The program can be done in several sections:

1. Generate random numbers within the range
2. Store results of up to 1000 tosses
3. Display output

The following code segment will generate the random numbers and convert them to integers within the range 1 and 2:

```
Randomise
for toss = 0 to 999
  HeadTails(toss) = Int (rnd(2))
next toss
```

This will fill the array called `HeadsTails` with either 1s or 2s.

Counting heads and tails

With 1000 numbers stored we can now scan the array and count the occurrences of 1s and 2s that represent heads and tails.

The following code segment should accomplish this:

```
For toss = 0 to 999
  If HeadsTails(toss) = 1 then
  heads = heads + 1
    else
  tails = tails + 1
  end if
next toss
```

Finally add the code to output the number of heads and tails.

A typical output is shown in Figure 7.16

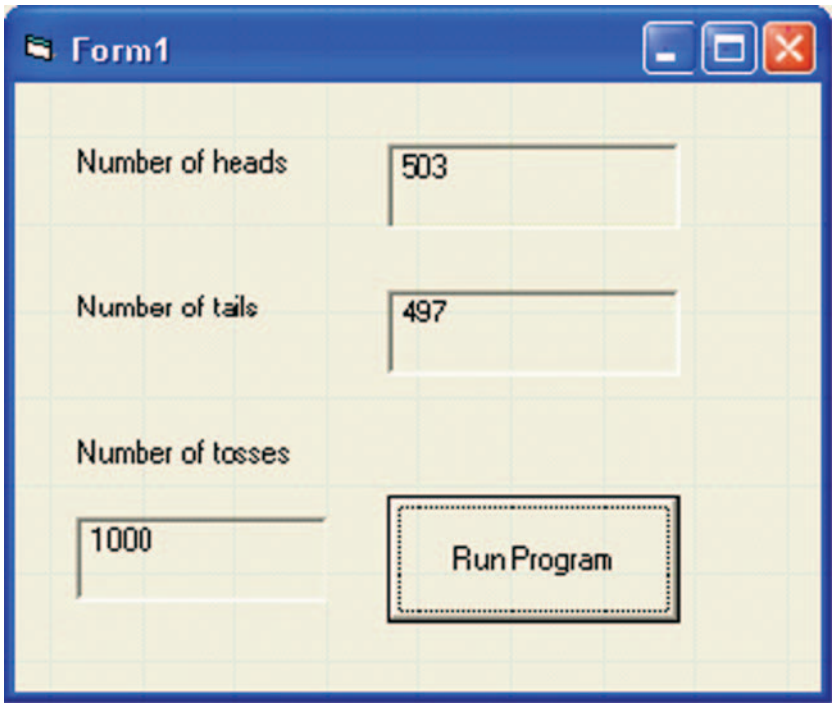


Figure 7.16:

Heads and Tails Program

Code the program and run it a few times to see if you get the expected result (i.e. 500 heads and 500 tails).

Dice Program

The program can easily be modified to simulate throwing a die.

Exercise

Amend Code 38 to output the results of throwing a die about 50 times.

7.8.8 Testing for Palindromes using an array

Problem: Write a program which will read in a sequence of words. The characters will be read into an array and the program should then determine if the sequence is a palindrome.

Note: A palindrome is a sequence of number/characters/words etc. which is the same when read from either direction.

The following examples are palindromes (if we ignore spaces), the sequence being the same when read from left to right or from right to left.

```
2 3 4 5 4 3 2

madam im adam

was it a car or a cat I saw

live not on evil

eve
```

This program may seem more complicated than it really is. If you follow the explanation you will see that by breaking it down into small steps, each of which can be written in Visual BASIC you will chip away at the problem until it is done.

The solution to this problem assumes that you do not know how many words are going to be input.

Design Solution

Notes on Section 2
The program does not know how big a sequence will be entered so an array of characters larger than required must be declared. The program reads in the sequence of characters one at a time into the array, terminated by a full stop. It also checks that the array bounds are not breached by testing the condition:

```
count <= Size
```

using a `do..while` loop.

This is important, otherwise an array bounds error will likely occur. The constant `Size` is initially set to 30 but may be altered if large strings are used.

The condition in the `while..do` loop that checks whether "." was the last character entered is

```
palindromeChar <> "."   '<> means 'not equal to'
```

Notes on section 3
Once it has completed reading in the characters, you can determine the number of values read in. This is `count - 1`, not count since you are not interested in the last character. All it contains is the "." termination character which you do not want to use in

the rest of the program.

Now you are ready to see whether the string is a palindrome. The array is searched through comparing the 1st element with the last, the 2nd element with the 2nd last etc. As soon as you find that they do not match you can terminate the search as you know that the sequence is not a palindrome. It is not necessary to continue searching the whole list of numbers as we know what we set out to find out.

Note that you only have to loop for `count <= length \ 2` as you compare the first half of the array to the second half. See following Figure 7.17:

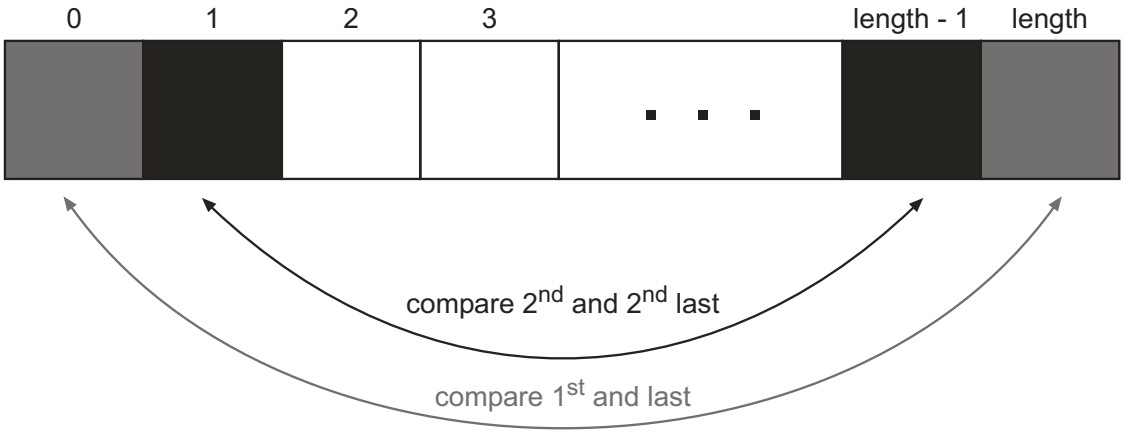


Figure 7.17: Comparing array elements for a palindrome

A possible solution is shown in Code 31.

```
Private Sub Command1_Click()
  'program Palindrome
  '20th February 2004
  'Program by Fred Fink
  'This program will test an input string for a palindrome

  Const sizeArray As Integer = 30

  Dim Palarray(1 To sizeArray) As String
  Dim palindromeChar As String
  Dim length As Integer, count As Integer
  Dim isPalindrome As Boolean
  count = 1

  Do While (palindromeChar <> ".") And (count <= sizeArray)
     palindromeChar = InputBox("Enter your characters,one at a time.
Terminate by full stop")
     Palarray(count) = palindromeChar
     PicDisplay.Print palindromeChar;   'Output characters
     count = count + 1
  Loop

  length = count - 1        'to account for increment when "." is
                             read in
```

```
    'Now check for palindrome

     isPalindrome = True
     count = 1
     Do While (isPalindrome) And (count <= length \ 2)
        If Palarray(count) <> Palarray(length - count) Then
        isPalindrome = False
        Else
        count = count + 1
        End If
     Loop

     If isPalindrome Then
       PicDisplay.Print ("  is a palindrome")
       Else
       PicDisplay.Print ("  is not a palindrome")
    End If

End Sub
```

This file (Palindrome.txt), can be downloaded from the course web site.

Code 31

Further notes on Section 2

Examine the `do..while` statements carefully. You will see that the conditions are compound.

- the character read in must **not** be a full stop AND there must still be space left in the array
- if both these conditions are fulfilled, then the loop continues
- if either one or the other condition fails, then the loop terminates.

Further notes on Section 3

The `do while (isPalindrome) and (count <= length \ 2)` statement tests whether you have gone halfway through the array AND whether the boolean variable '`isPalindrome`' has been set to false.

This needs a bit more explanation:

- notice that we initialised `isPalindrome` to `true` at the beginning of the program
- if you look at the output statements right at the bottom of the code, you can see that if `isPalindrome` is still true, then the array is a palindrome
- if `isPalindrome` has been set to `false` at any point, then the array is not a palindrome

- this is what a boolean *flag* is for. You set it to a state - here it is either `true` or `false` - and you use this value to see whether a certain condition still holds true
- so what might set `isPalindrome` to `false`? It is the code inside the `while..do` loop which compares the first and last items
- if they are *not* the same, the boolean flag is set to `false`. Otherwise nothing happens
- then the second, and second from last are compared..... and so on.

In terms of loop tests this is probably about as complicated as it gets. If you do not understand it this time around, just be patient - you will probably find that you need to code something like this yourself in the future and you can come back to this example. By actually doing it, the difficulties seem much less than by reading it as you are doing now.

Coding the Palindrome Checker

Now code and test the palindrome checker. You can save yourself time by copying and pasting the code into your VB editor, once you have designed the interface and form.

Improved palindrome checker

Adapt the previous program to ignore any spaces and any upper / lower case differences.

Comparing arrays and encoding the results

30 min

Declare three 20-element arrays, X, Y, Z. Write a program to read 20 integers into each of the two integer arrays `X` and `Y`. The program will prompt the user to enter the values into each of the arrays. The program will then compare each of the elements of `X` to the corresponding element in `Y`. Then, in the corresponding element of a third array `Z`, store the following values, Table 7.2.

Table 7.2

`Z` **Element Value**	**Condition**
1	if the element in `X` is larger than the element in `Y`
0	if the element in `X` is equal to the element in `Y`
-1	if the element in `X` is less than the element in `Y`

Then print out a three column table displaying the contents of the arrays `X`, `Y` and `Z`. Make up your own test data and write down in three columns the number you input for X, the number you input for Y and the result you got for Z

7.9 Summary

The following summary points are related to the learning objectives in the topic introduction:

- understand and be able to use the ’for ..next’ structure;
- understand and be able to use the ’do..while’ structure and variant;
- understand and be able to use the do..until structure and variant;
- how to declare 1-D arrays;
- initialise a 1-D array;
- manipulate data held in 1-D arrays.

7.10 End of topic test

An online assessment is provided to help you review this topic.

Topic 8

Procedures, Functions and Standard Algorithms

Contents

Prerequisite knowledge

Before studying this topic you should be able to:

- *describe and exemplify pre-defined functions.*
- *recognise appropriate use of the following standard algorithms:*
  - *input validation;*
  - *find min/max;*
  - *count occurrences;*
  - *linear search.*

Learning Objectives

After completing this topic, you should be able to:

- *understand and use procedures in programs*
- *understand and use functions and user-defined functions in programs*
- *describe how the use of procedures and functions aids modularity of programs*
- *understand parameters and how they are passed (in,out,in/out)*
- *understand parameter call by value and call by reference*
- *be able to describe in pseudocode and implement:*
  - *standard algorithms,*
  - *linear search, counting occurrences and finding maximum/minimum.*

Revision

Q1: A piece of programming code contains a validation routine. This is to ensure:

a) That the program produces the correct output
b) That the input data is within specified limits
c) That the output is within specified limits
d) That the input data is restricted to characters only

Q2: Programming languages usually contain a collection of pre-defined functions. Which one of the following statements is true?

a) Programming time can be saved
b) Functions produce a single value
c) Functions can be used with or without parameters
d) All of the above

Q3: What is meant by the term standard algorithm?

a) Universal code to solve all problems
b) A sequence of instructions that can be used to solve a common problem
c) Code that makes a program more reliable
d) Any program written in a high level language

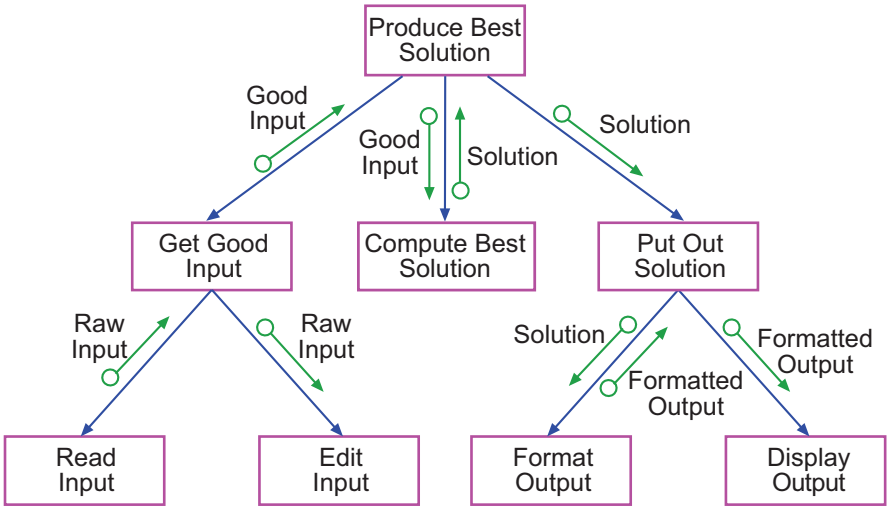
8.1 Introduction to Modular programming

This final topic introduces further modular programming concepts. Programs are built up from individual blocks of code - procedures and functions.. The main program will then consist of a series of procedure/function calls which makes any program more readable and easier to maintain.

The fairly complex issue of parameters and parameter passing is covered in detail and exemplified by working solutions.

Finally three standard algorithms required for Higher Software Development are discussed with exemplar code for each supplied.

You will recall from Topic 3 that an important aspect of the design phase was the idea of refining a complex program by progressively breaking it down into smaller, easier-to-solve units *(top-down design with step-wise refinement)*. Each of these units, called modules or program blocks, can be independently coded in a high level language and then amalgamated to form the complete program. The use of structure charts emphasise this modular design and also the relationship between all the modules in the software system.



In comparison to commercial software the programs that you will be writing in Visual Basic will be relatively short and you might not see the need to use modular techniques. The Windows operating system, for example contains well over 30 million lines of code. Just think of trying to program this in one large block of code! This would be an impossible task without the system being decomposed into many smaller, discrete units that can be easily modified and more readily debugged and compiled. Your own programs, however small they may be, should embody the principles of modularity.

Visual Basic is an event driven programming language where the programming is done in a graphical environment. Users can click on various objects where each object is programmed independently to be able to respond to user actions. A Visual Basic program is, therefore, modular, being made up of many subprograms, each having its own program code that can be executed independently.

8.2 Procedures and Functions

Procedures and **functions** provide a means of producing structured programs, not only in Visual BASIC but in other programming languages as well. Procedures and functions allow a program to be broken up into into more meaningful and logical sections.

Advantages of modular programming:

- The repetition of lines of code is avoided. Rather than repeating the same operations several times in a program, the code can be placed in a procedure or function. The procedure is then called as many times as necessary without re-writing the code.
- It allows testing of the procedure code to take place in isolation from the main program. Each functional unit can be written independently by programming teams, making this part of the software development process more efficient.
- Debugging of the main body of the program is simplified since each procedure can be individually tested.
- Procedure code can be saved and re-used in future projects. Module libraries,

for instance are repositories for useful chunks of code that can be accessed by programmers who could save time by using such code, instead of 're-inventing the wheel'.

Procedures and functions are somewhat similar in their structure. They both consist of:

- a heading
- a declaration part (where necessary)
- an action part (a compound statement)

A typical Procedure

```
Sub getValidNumber   (ByRef number As integer)
Dim numberOk As boolean
  do
    Print ("Input a number ")
    numberOk = number >= 0
    if not numberOk then
      print ("The number must be greater than 0")
    end if
  loop until numberOk
End sub
```

A typical Function

```
function getValidNumber As integer
Dim numberOk As boolean
Dim number As Integer
  do
    print ("Input a number ")
    numberOk = number >= 0
    if not numberOk then
      print ("The number must be greater than 0")
    end if
  loop until numberOk
  getValidNumber = number
end function
```

A procedure or function is activated by a call from the main program, after which the procedure or function executes its block of code and then terminates.

The flow of data between procedures, functions and the main program block is accomplished by the use of **parameters**, which will be discussed later.

8.2.1 Procedures

Visual BASIC offers a variety of procedure types. The two that concern us here are:

- event procedures
- general procedures.

Up to now most of the programming code you have seen has been made up of *event procedures* that activate sections of code when, say a command button is pressed. These procedures are named by Visual BASIC by concatenating the name of the object code and the name of the event according to the syntax:

```
Sub Command_Click()
  End
End Sub
```

You will recognise this code which ends a Visual BASIC program.

8.2.1.1 General Procedures

The basic structure for a general procedure is:

```
Sub ProcedureName (formal parameters)
Declarations
Statements
End sub
```

Notes

1. The name, `ProcedureName` or identifier, is what is used when the procedure is called and the name conforms to the same rules of naming variables.
2. Data is passed to and from the main program using the procedure `parameters` or `arguments` enclosed in parentheses. Not all procedures however need to have parameters.
3. The *declarations* and *statements* follow an identical pattern to normal programming constructs.
4. Procedures are normally declared `private` and their scope is limited to other procedures and variables within the current form.

Here, to get us started is a simple little procedure that prints blank lines on a form. This might be useful for putting spaces into program output.

```
Private Sub BlankLines
  Print
  Print
  Print
End Sub
```

This could be called, in the action part of the main block, as:

```
Call BlankLines
```

Its use in a program could look like this:

```
Private Sub command1_Click()
  'program lines
  Print("Message 1.......")
  Call BlankLines
  Print("Message 2.......")
End Sub

Private Sub BlankLines
  Print
  Print
  Print
End Sub
```

As it stands, `BlankLines` isn't very well structured. As programmers, we should always be a little wary when we find ourselves repeating code. It might be better to put the `Print` statements within a loop structure. The following refinement should produce a more acceptable program:

```
Private Sub BlankLines
  for n = 1 to 3
    Print
  Next n
End Sub
```

The program is still not very flexible. We should be able to tell the procedure how many blank lines to output.

This is achieved using **parameters** and we will come back to this section of code later.

Program with parameters

A user is asked to input numerical values between 1 and 30. Write a section of code as a procedure to validate user input between these two values. Show how it would be called from the main program.

8.2.1.2 Parameter passing

How are parameters used to pass information between sub-programs?

To see how this works, consider the following program:

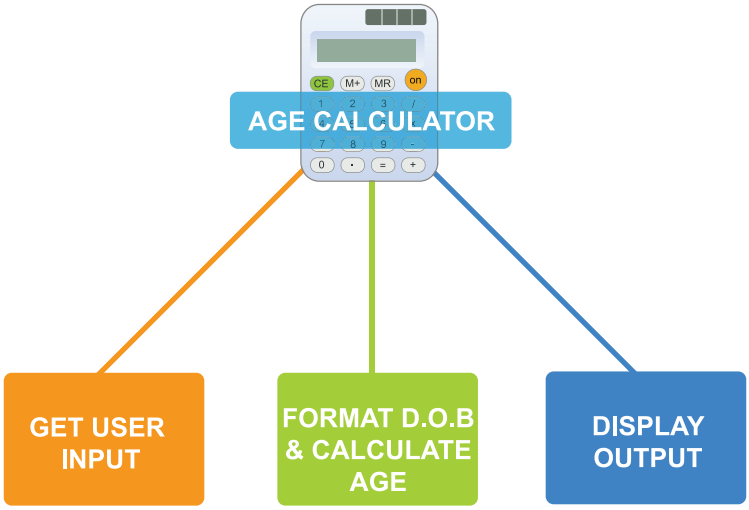
Specification:
The program must ask a user to enter their name and date of birth. It should then convert

the date of birth into standard format ddmmyyyy, and calculate the person's age. The name, standard format d.o.b. and age should then be displayed on the screen.

Design:
There are 3 clear stages. In pseudocode, these could be written as:

1. get user input
2. format d.o.b and calculate age
3. display output

As a **structure diagram**:



Each of the three stages can be coded as a procedure sub-program.

Implementation:

The coding for this would be as follows (omitting details):

```
Private Sub cmdAge_Click()
  'comment lines
  Call Get_user_input
  Call Do_calculations
  Call Display_output
End Sub

Private Sub Get_user_input
  .....
End Sub
Private Sub Do_calculations
  .....
End Sub
Private Sub Display_output
  .....
```

```
End Sub
```

Parameters:

As it stands, this program could not work, as data items cannot be transferred between the sub-programs. To be able to work, we need a mechanism to transfer data items between the 3 sub-programs. This is achieved using parameters.

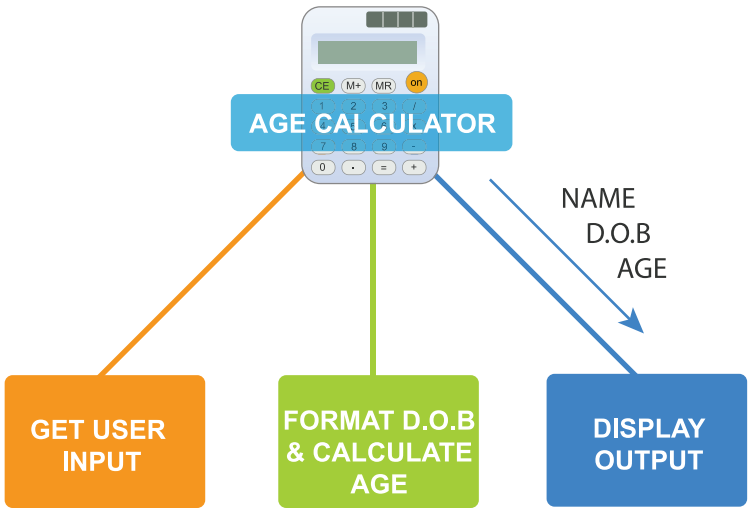
First, think about the last procedure, Display_output.

To do its work, it needs to know:

- the user's name
- the user's d.o.b.
- the user's age

These data items must be passed IN to this procedure.

We can show this on the structure diagram (and the pseudocode):



1. get user input
2. do calculations
3. display output IN: name, dob, age

Next, think about the middle procedure, Do_calulations.

Its output will be:

- the user's d.o.b. (formatted correctly)
- the user's age

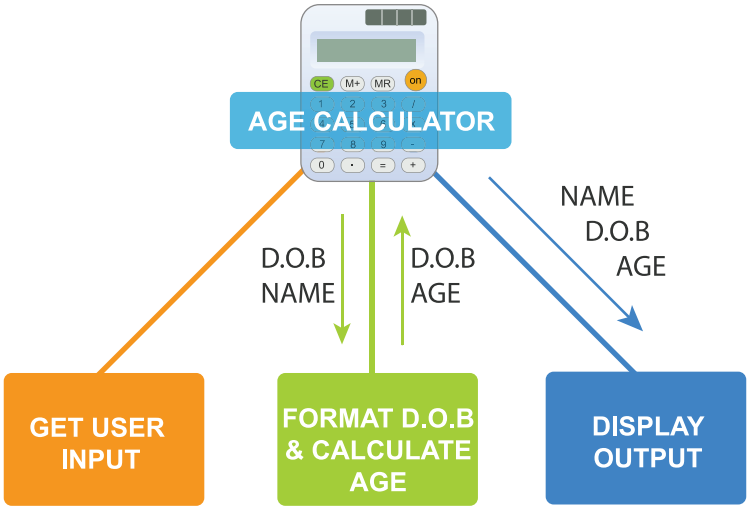
These data items will be passes OUT of the procedure.

To produce these outputs, it needs to know:

- the user's name
- the user's d.o.b. (as supplied by the user)

These data items must be passed IN to this procedure.

We can add this to the structure diagram (and the pseudocode):



1. get user input
2. do calculations IN: name, dob OUT: dob, age
3. display output IN: name, dob, age

Finally, think about the first procedure, Get_user_input.

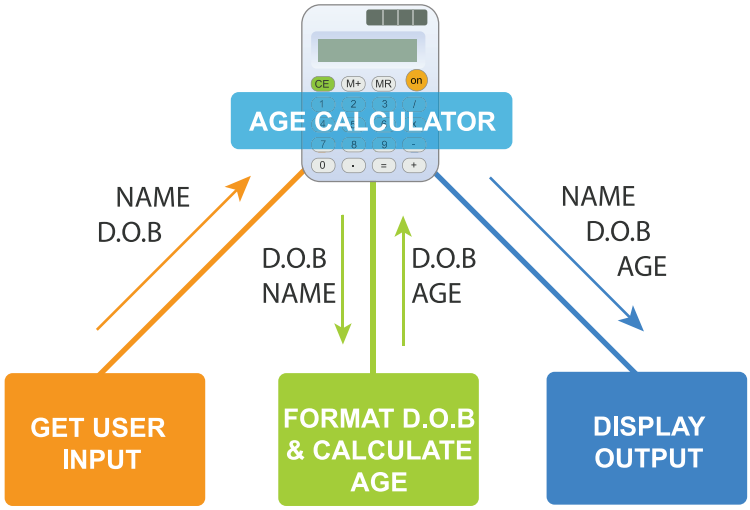
Its output will be:

- the user's d.o.b.
- the user's name

These data items will be passes OUT of the procedure.

It receives its input directly from the user when it is executed.

We can now complete the structure diagram (and the pseudocode):



1. get user input OUT: name, dob
2. do calculations IN: name, dob OUT: dob, age
3. display output IN: name, dob, age

All of these data items listed above, i.e.

- name
- d.o.b.
- age

will need to be **parameters**.

8.2.1.3 byVal and byRef

Visual BASIC, like most other programming languages, allows two types of parameter passing; in VB, these are called **byVal** and **byRef**.

byVal (call by **Value**) is used when a parameter is only an input to a sub-program.

byRef (call by **Reference**) is used when a parameter is an output from a sub-program.

Some parameters are both IN and OUT (e.g. dob is both in and out for the do_calcualtions procedure). In that case, the parameter must be passed **byRef**.

Now we can complete the coding for the program, showing these parameters:

- in the main program, we declare all the parameters using Dim
- in the main program, after each sub-program call, we list all the parameters used by that sub-program (whether in our out) - these are called the **actual parameters**
- after the header of each sub-program, we list its parameters again, indicating for each one, whether it is to be passed by value or by reference - these are called the **formal parameters**

```
Private Sub cmdAge_Click()
  'comment lines
  Dim name as String, dob as String
  Dim age as Integer
  Call Get_user_input(name,dob)
  Call Do_calculations (name, dob, age)
  Call Display_output (name, dob, age)
End Sub

Private Sub Get_user_input (byRef name as String, byRef dob as String)
  .....
End Sub

Private Sub Do_calculations (byRef name as String,byRef dob as String, byRef age as
               integer)
  .....
End Sub
Private Sub Display_output (byVal name as String, byVal dob as String, byVal age as
               integer)
  .....
End Sub
```

Note: it is important that the order of the actual parameters (in the main program) matches the order of the formal parameters in a sub_program. However, the names can be different!

For example:

Private Sub Do_calculations (byRef name as String, byRef age as integer, byRef dob as String) is WRONG, because the parameters are listed in the wrong order.

but

Private Sub Do_calculations (byRef x as String, byRef y as String, byRef z as integer) would be fine, because, although the names are changed, the parameters are listed in the correct order, so VB would know that *x* corresponds to name, *y* to dob and *z* to age.

You can use the same names for both formal and actual parameters if you want to. However, using different names allows a sub-program to be included in a module library, and used in different contexts without having to change the variable names within the procedure.

byVal or byRef

Consider each of the following situations and decide whether the parameter needs to be called/passed by Ref or by Val.

Q4: Interpreting mouse movements

a) byRef
b) byVal

Q5: Checking there is sufficient RAM to load a program

a) byRef
b) byVal

Q6: Keeping track of where documents are stored

a) byRef
b) byVal

Q7: Sending data to a printer

a) byRef
b) byVal

8.2.1.4 Call by Value

Call by value is the most common parameter passing mechanism and is also the easiest to understand. It is used when a data item is to be passed IN to a procedure.

Recalling the section of code that outputs blank lines we can now introduce a **value parameter** to make the procedure much more flexible.

Here is the Visual BASIC code (Code 32)

```
Private Sub Command_Click()
  'Modified program lines
  Print("Message 1.......")
```

```
  Call BlankLines(5)
  Print("Message 1.......")
End Sub

Private Sub BlankLines(ByVal Number as Integer)
  Dim count As Integer
  For count = 1 To Number)
    Print
  Next Count
End Sub
```

Code 32

The actual parameter is 5; the formal parameter is Number. By calling Blanklines(5), the actual parameter (5) is passed into the formal parameter Number in the procedure. The procedure code is executed with Number = 5, then it terminates.

Note that the formal parameter is preceded by the Visual BASIC keyword `ByVal` to indicate a value parameter. It is essential to declare whether the parameter is called by value or by reference.

The following worked example will help you understand passing by value:

Example 1: Use of procedures with value parameters

Problem: Write a program that outputs the square and square root of a number that is input by the user. This number will be passed to each of the procedures.

Solution

The following algorithm uses two procedures with *value* parameters:

```
1 set up variables
2 get valid user input
3 call subprogram "square"
4 call subprogram "sqrRoot"
```

The full Visual BASIC program is shown in Code 33.

```
Option Explicit

Private Sub Command1_Click()

   'Main Program

   'Program by Fred Fink
   '25th February 2004

   'This program exemplifies the use of value parameters
```

```
   Dim ActualValue As Integer

   Do
     ActualValue = InputBox("Input a value between 1 and 100")
   Loop Until (ActualValue >= 1) And (ActualValue <= 100)

   Call Square(ActualValue)              'Call procedure
   PicDisplay.Print
   Call SqrRoot(ActualValue)             'Call procedure

End Sub

Private Sub SqrRoot(ByVal number As Integer)   'Procedure declaration
   Dim Result As Single
   Result = Sqr(number)
   PicDisplay.Print "The square root of "; number; " is "; Format(Result,
                                                              "Fixed")
End Sub

Private Sub Square(ByVal number As Integer)    'Procedure declaration
   Dim Result As Integer
   Result = number ^ 2
   PicDisplay.Print "The square of "; number; " is "; Result
End Sub

Private Sub Command2_Click()
End
End Sub
```

This file (ValueParameter.txt), can be downloaded from the course web site.

Code 33

Squares and square roots with procedures

Examine this code carefully. Make sure you understand its structure. The fixed format command is to output the square root to two decimal places.

Program output is shown in Figure 8.1

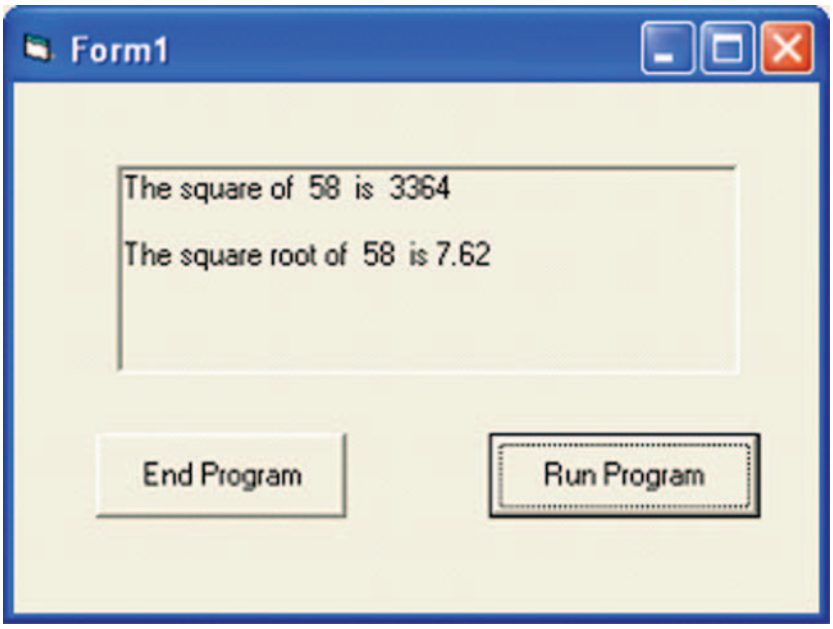


Figure 8.1:

Note that the original value of the variable `ActualValue` in the main program remains unchanged. Any changes made by the procedure to the value passed are local to the procedure and not *passed back* to the main program. In other words the procedure can only modify a *copy* of the variable value and not the variable `ActualValue` itself. You will see more of this later.

In a procedure call the word `call` may be omitted with only the procedure name and parameters required. For example:

`Call Square(ActualValue)` becomes Square `(ActualValue)`

This notation will be used from now on in all programs.

One of the disadvantages of call by value is that a copy of the actual parameter is always made in order to produce the formal parameter during the procedure call.

8.2.1.5 Call by Reference

Call by reference is used when information has to be passed **out** from a procedure to the main program. Here the variables inside the procedure body are allowed to reference the memory location of the actual variable that passed it. This means that as the actual parameters change as do the formal parameters.

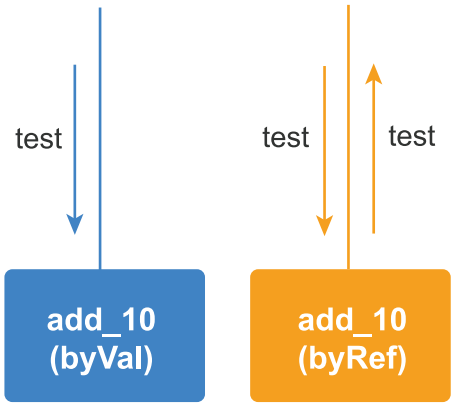
A reference parameter does not pass a value of a variable but instead passes the address of the variable.

It is important that you understand the difference between value and reference parameters.

Consider the following procedures. They are identical, but one uses call by value and the other uses call by reference.

```
Private Sub Add_10(ByVal Number As Integer)
  Number = Number + 10
End Sub

Private Sub Add_10(ByRef Number As Integer)
  Number = Number + 10
End Sub
```

If they are now executed within a program we can see the difference to the variable Test in the main program by calling each procedure.



The full Visual BASIC program is shown in Code 34

```
Option Explicit

Private Sub Command1_Click()
  Dim Test As Integer
  Test = 10
  PicDisp1.Print Test
  Add_10(Test)
  PicDisp2.Print Test
End Sub

Private Sub Add_10(ByVal Number As Integer)
  Number = Number + 10
End Sub
```

Code 34

Comparing byVal and byRef

1. Run the program as above.
2. change "byVal" to "byRef"
3. Run the program again.

Compare the output. Can you see why this is happening?

Your results should appear something like this:

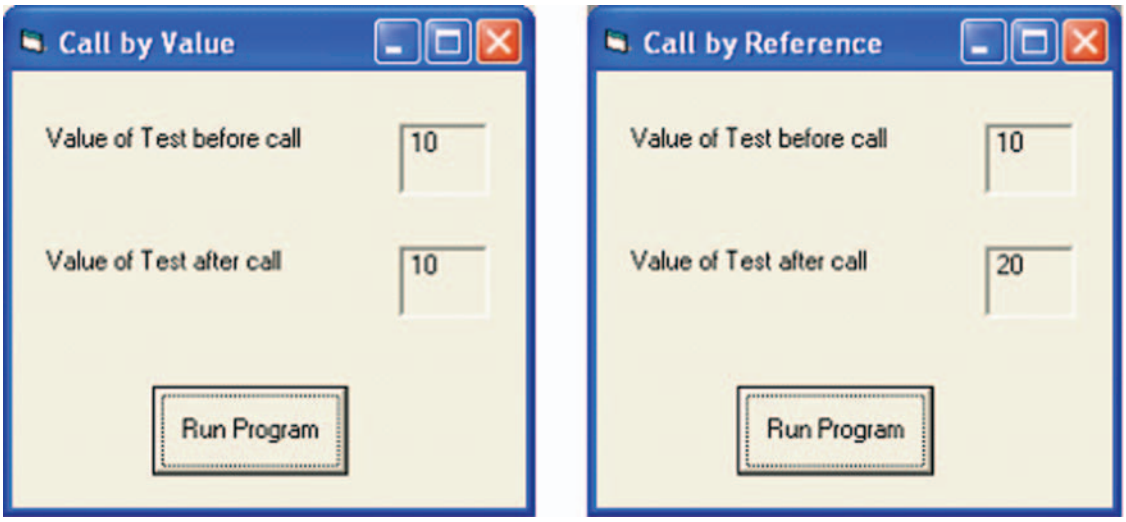


Figure 8.2:

Each procedure call produces the correct value of 20 for `Test` but only in call by reference is this value passed back to the main program. In call by value the main program knows nothing about the local changes made within the procedure.

The process of events is as follows:

1. The procedure call `Change(test)` is initiated;
2. The procedure declaration `Change` is accessed;
3. Procedure `Change` now accesses the memory location of variable `Test` and gets the numerical value 10. Formal parameter `number` now has the value 10.
4. Procedure `Change` code is activated and both `number` and `Test = 20`.

8.2.1.6 Comparison of Call by Value and Call by Reference

How do Call by Value and Call by Reference work? If you have access to the web version of SCHOLAR, you should view the animated version of the following explanation. Consider the previous program.

If Call by Value is used, the process is:

1. The Procedure call Add_10(Test) is initiated.
2. The Procedure declaration Add_10 is accessed.

3. Within the procedure, a local copy of the variable Test is made, named Number.
4. The Procedure adds 10 to the value stored in Number.
5. The procedure terminates.
6. Control returns to the main program.
   Test still has the value 10.

If Call by Reference is used, the process is:

1. The Procedure call Add_10(Test) is initiated.
2. The Procedure declaration Add_10 is accessed.
3. The Procedure accesses the main program variable Test directly, and treats it as though it is the local variable Number.
4. The Procedure adds 10 to the value stored in Test.
5. The procedure terminates.
6. Control returns to the main program.
   Test now has the value 20.

Don't worry if you have found the material in this section difficult to follow first time round.

Remember!

- if a parameter is used only to transmit a *value* **to** a procedure then make it a value parameter
- if a parameter represents a result that is produced **by** the procedure to be used elsewhere in the program then make it a reference parameter.

Note: Call by reference is the default condition for Visual BASIC, but for Visual BASIC.Net users the default condition is call by value.

8.2.1.7 Program using reference parameters

Problem: A user enters two numbers at the keyboard. Write a program that swaps the positions of the numbers.

Design

The program will use 3 procedures, as shown in the pseudocode:

```
1 GetNumbers(Number1, Number2)
2 BlankLines(2)
3 swap(Number1, Number2)
```

Notice that we are using the procedure `BlankLines`, the first procedure we met at the start.

Designing the solution

1. Draw a structure diagram.
2. Show, by arrows, the flow of data in and out of the procedures getNumbers and Swap.
3. Complete the pseudocode, showing the IN and OUT parameters for each procedure.

Implementation

The full Visual BASIC program is shown in Code 35

```
Option Explicit

Private Sub Exchange_Click()
  Dim Number1 As Integer, Number2 As Integer

  GetNumbers (Number1, Number2)
  BlankLines(2)
  swap (Number1, Number2)

End Sub

Private Sub BlankLines(ByVal Num As Integer)
  Dim count As Integer
  For count = 1 To Num
    Display.Print
  Next count
End Sub

Private Sub swap(ByRef value1 As Integer, ByRef value2 As Integer)
  Dim temp As Integer
  Display.Print ("Numbers before swap = "); Tab(25); value1; " ";value2
  temp = value1
  value1 = value2
  value2 = temp
  BlankLines(1)
  Display.Print ("Numbers after swap = "); Tab(25); value1; " "; value2;
End Sub

Private Sub GetNumbers(ByRef Num1 As Integer, ByRef Num2 As Integer)
  Num1 = InputBox("Input first value")
  Num2 = InputBox("Input second value")
End Sub

Private Sub cmd_Exit_Click()
  End
End Sub
```

This file (RefParameter.txt), can be downloaded from the course web site.

Code 35

The first call of procedure `GetNumbers` produces the values for formal parameters `Num1`

and `Num2` through user input. These values are passed to the actual parameters `Number1` and `Number2`.

`BlankLines(2)` produces two clear lines before output of results.

The values are now swapped:

```
temp = 20 value1 = 40 value2 = temp
```

Notice the procedure `BlankLines(1)` is nested within procedure `GetNumbers`.

Results are displayed via procedure `GetNumbers`.

Typical output is shown in Figure 8.3

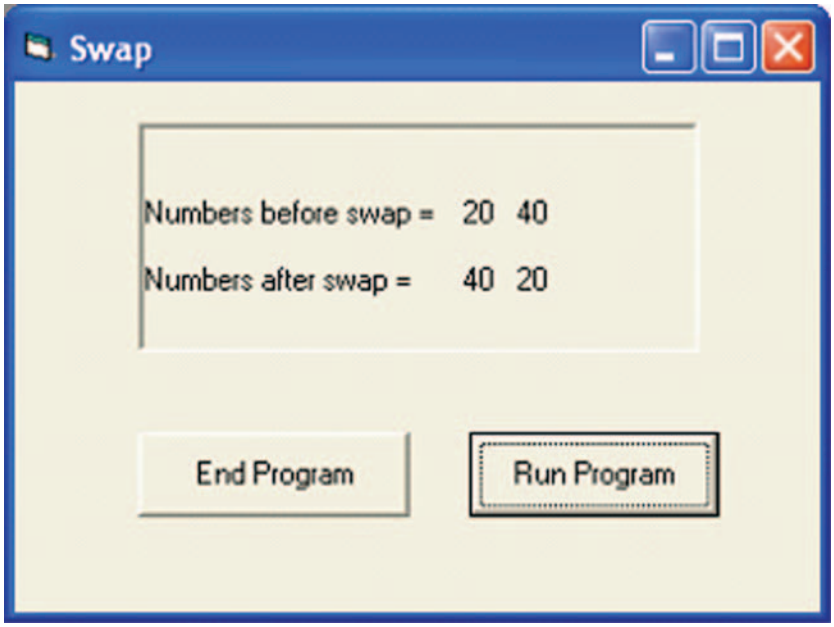


Figure 8.3:

8.2.1.8 Review Questions

Q8: The use of procedures in a program offers many advantages to the programmer. Which one of the following would represent such an advantage?

a) They must have meaningful names
b) Repetition of lines of code are avoided
c) Procedures can be nested within procedures
d) They can work with or without parameters

Q9: The structure of a procedure consists of three main parts. Which one of the following is not one of these parts?

a) A heading
b) A declaration
c) An action
d) A footer

Q10: Parameters can be classified as being IN, OUT and IN-OUT. Which one of the following statements is true regarding these parameters?

a) byVal is used for IN-OUT parameters
b) byRef is used for IN-OUT parameters
c) byRef can only be used for IN parameters
d) byVal can only be used for OUT parameters

Q11: In a procedure where a parameter is called by reference, which one of the following is true?

a) A reference parameter is an example of an IN parameter.
b) Anything that happens to the formal parameter happens to the actual parameter.
c) The parameter must be an Integer variable.
d) Only one parameter can be called by reference.

Q12: What would be the output of the following Visual BASIC code?

```
Dim x As Integer, y As Integer
  x = 5 y = 6
    Print x, y
    result (x, y)
    Print x, y
  Sub result(ByRef num1 As Integer, ByVal num2 As Integer)
    num1 = num1 + 10
    num2 = num2 - 3
  End Sub
```

a) 5 6 5 3
b) 5 6 3 5
c) 5 6 15 6
d) 5 6 15 3

8.3 Functions

A function is similar to a procedure, except it returns a single value to the calling code. The basic structure for a function is:

```
Function functionName (parameters) As Data Type
  Declarations
  functionName = expression
End Function
```

Since a function returns a value it is used within expressions. The function name must, in the action part of the function, be assigned the value that is to be returned. A function is called using a statement of the form `variable = functionName(parameters)`

8.3.1 Pre-defined functions

Some built-in Visual BASIC functions have been used in many of the programming examples up to now. Table 8.1 shows some of the more common pre-defined functions:

Table 8.1:

Name	Function	Example
ABS(X)	Returns the absolute value of X	ABS(5.6) = 5
INT(X)	Returns truncated integer part of X	INT(34.5) = 34
SQR(X)	Returns the square root of X	SQR(25) = 5
RND	Generates a random number between 0 and 1	X = (100*RND) will give random numbers from 1 to 100
TAB(X)	Outputs at print position 'X'	Print TAB(5)
SPC(X)	Outputs number of spaces between last printed position and the next	Print SPC(3)
ASC("X")	Returns the ASCII value of a character "X"	Print ASC("A") returns 65
CHR$(X)	Returns an ASCII value into a character	Print CHR$(68) returns "D"
VAL("X")	Returns the value of string "X"	VAL("76923") = 76923
STR$(X)	Returns the string of value X	STR$(1234) = "1234"

8.3.2 User-defined functions

User-defined functions work in exactly the same way as Visual BASIC a pre-defined functions and extend the range of programming possibilities.

Here, as a simple example, is a little function that simply doubles the value sent to it.

```
function double (number as integer) as integer
  double = number * 2
end function
```

The function would then be called as

```
new_value = double(10)
```

We could combine this in a section of a program with a procedure to get a valid (positive) number:

```
'Program getNumbers
  Dim number integer
  getValidNumber (number)
  print ("Twice that is ", double (number))

sub procedure getValidNumber (ByRef number as integer)
  do
```

```
    number = inputbox("Enter a number: "
  loop until (number > 0)
end sub

function double (number as integer)as integer
  double = number * 2
end function
```

Simple Doubler

Code the above section of code into a Visual BASIC program and run it with different values.

8.3.3 User-defined Function Examples

Example 1: Area of a Circle:
A program has to be written that uses a function to output the area of a circle, given the radius.

Solution

We will use the following algorithm:

```
1 display table headers
2 For each radius
3 Print radius and area
4 Next radius
```

The Visual BASIC program is seen in Code 36

```
Option Explicit
Const Pi As Single = 3.141592
Private Sub Command1_Click()
  Dim Radius As Integer
  PicResult.Print Tab(4); "Radius"; Tab(16); "Area"
  PicResult.Print
  For Radius = 1 To 10
    PicResult.Print Tab(6); Radius; Tab(16); Format(Area(Radius),  "Fixed")
  Next Radius
End Sub
Function Area(Radius) As Single
  'This function will calculate area of circle
  Area = Pi * Radius ^ 2
End Function

Private Sub EndProgram_Click()
  End
End Sub
```

This file (FunctionArea.txt), can be downloaded from the course web site.

Code 36

Turning a procedure into a function

Any procedure that returns a single reference parameter cans be re-written as a function.
Rewrite this procedure as a function. It will return a valid integer. It will no longer need a parameter.

```
sub getValidNumber(ByRef number as integer)
Dim numberOK as Boolean
  do
    number = Inputbox("Input a number")
    numberOK = number > 0
    if not numberOK then
      Print("The number must be greater than 0")
    end if
    loop until numberOK
End sub
```

8.4 Reviewing Functions and Procedures

Q13: Which one of the following statements regarding a function is **true**?

a) A function does not need to be declared.
b) A function cannot be defined by a user.
c) A function returns a single value.
d) Any procedure can be rewritten as a function.

Q14: Below are four functions. Which one is not available as a predefined function in Visual BASIC?

a) Inputbox
b) Val
c) Square
d) Rnd

Q15: Look at the Visual BASIC statements below that use built-in functions. Which one represents a valid function call?

a) SQR(x) = x
b) y = ABS(-5)
c) 65 = CHR$("A")
d) z = SQR("z")

Function or Procedure?

You should now try the on-line activity which tests your understanding of when it is appropriate to use a function or a procedure with parameters.

8.5 Standard Algorithms

An algorithm is a finite sequence of steps which, when followed, will accomplish a particular task.

The term algorithm derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi (c.825AD) who was a mathematician and part of the royal court in Baghdad. Al-Khwarizmi's work is the likely source for the word *algebra* as well.

Figure 8.4: Cunieform tablet

There is ample proof that the use of 'algorithms' was evident around 2000 - 3000 BC by the ancient Sumerians whose work was inscribed on clay tablets using a cunieform cipher.

Translating this text revealed mathematical rules and astronomical data written in sexagesimal notation (base 60) from which remains 360 degrees for circular measure, 60 minutes per hour, 60 seconds per minute and so on.

A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a procedure that solves a problem.

One major objective of this topic is to introduce you to common algorithms that have been tried and tested by programmers with knowledge of good algorithm design.

Many algorithms appear over and over again, in program after program. These are called **standard algorithms** or *common* algorithms.

Think about a word processing package which uses an algorithm to find all occurrences of a particular word in a block of text, or a spreadsheet package which uses an algorithm to find the maximum value in a range of cells. These packages make use of standard algorithms and it is worthwhile for every programmer to know them. When implemented these standard algorithms may become key components in a module library.

This section introduces 3 common algorithms used by programmers. Namely:

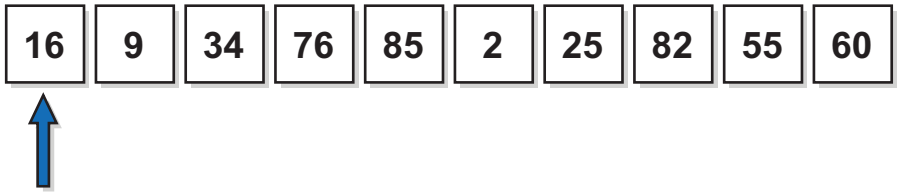
1. linear search
2. counting occurrences
3. finding maxima and minima.

You need to have prior understanding of arrays, as examples given rely upon knowledge of accessing and manipulating such structures.

8.5.1 Linear Search

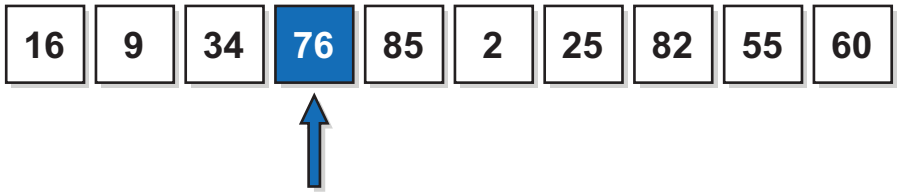
One task which computers frequently perform is to search through lists. For example, looking up a telephone number in a database, finding and replacing a word in a text file or looking for a particular stock item in a warehouse etc.

Linear search is the simplest search method to implement and understand. Starting with an array holding say, 10 numbers with a pointer indicating the first item the user inputs the item being searched for, which is known as the search key. Scanning then takes place from left to right until the **search key** is found, if it exists in the list. Look at the list below:



Suppose the search key is 76.

1. 16 is compared to 76. Not the key so pointer moves on one place
2. 9 is compared to the key. Not equal so pointer moves on
3. 34 is compared to key. Not equal so pointer moves on
4. 76 compared with key. Success! Key found at position 4 in the list.



8.5.2 Linear Search Examples

There are (at least) two variants of the linear search algorithm. The simplest searches the entire list from beginning to end.

Linear search 1 - searching the entire list

```
1. prompt the user for the search key
2. start at the first element in the list
3. do while not end of list
4. if the current element <> search key then
5. move onto the next element in the list
6. else output search key and position
7. end if
8. loop
```

This is not a very efficient way of doing things. Why bother to search the remainder of a list when the item has been found? There are instances when you **will** want to search

the entire list e.g. if you wish to replace **all** words in a list that begin with 'L' to words that begin with 'P' then the search would stop when you reach the end of the list.

The alternative algorithm makes use of a complex condition and a boolean variable:

Linear search 2 - stopping when the search key is found

```
1.  set found to false
2.  prompt the user for the search key
3.  start at the first element in the list
4.  do while (not end of list) and (found is false)
5.  if the current element = search key then
6.  set found to true
7.  display message
8.  else
9.  move onto the next element in the list
10. end if
11. loop
12. if not found then
13. display message
14. end if
```

The full Visual BASIC program is shown in Code 37. Examine it carefully to see how it operates, based on the algorithm. The output of this can be seen in Figure 8.5

```
Option Explicit
Dim Found As Boolean
Dim SearchKey As Integer, Pointer As Integer, Fill As Integer
Dim List(15) As Variant
Private Sub Command1_Click()

'Program Linear_Search
'29th February 2004
'Program by Fred Fink

'This program will perform a linear search on
'an array holding 16 random integers between
'the values 1 and 99

Setup                                          'Call procedures
Populate_List List()
Enter_search_item SearchKey
Search_for_item Pointer, SearchKey, List()

End Sub


Private Sub Setup()                            'Initialise variables
  Randomize                                    'and clear output boxes
  Found = False
```

```
  PicList.Cls
  PicResult.Cls
  PicResult.Print
  Pointer = 0
End Sub

Private Sub Populate_List(ByRef List())        'Fill array with
  Dim Fill As Integer                          '16 random numbers
  For Fill = 0 To 15
    List(Fill) = Int(99 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Enter_search_item(ByRef SearchKey) 'User input search item
  SearchKey = InputBox("Input search key")
End Sub

Private Sub Search_for_item(ByVal Pointer, ByVal SearchKey, ByRef List())
  Do While (Pointer <= 15) And (Not Found)

         If List(Pointer) = SearchKey Then
            Found = True
            PicResult.Print "Search item "; SearchKey; " found at position
                            "; Pointer
         Else
           Pointer = Pointer + 1
         End If
  Loop

  If (Not Found) Then
    PicResult.Print "Search item"; SearchKey; "is not in list!"
  End If
End Sub

Private Sub Command2_Click()
  End
End Sub
```

This file (Linear.txt), can be downloaded from the course web site.

Code 37

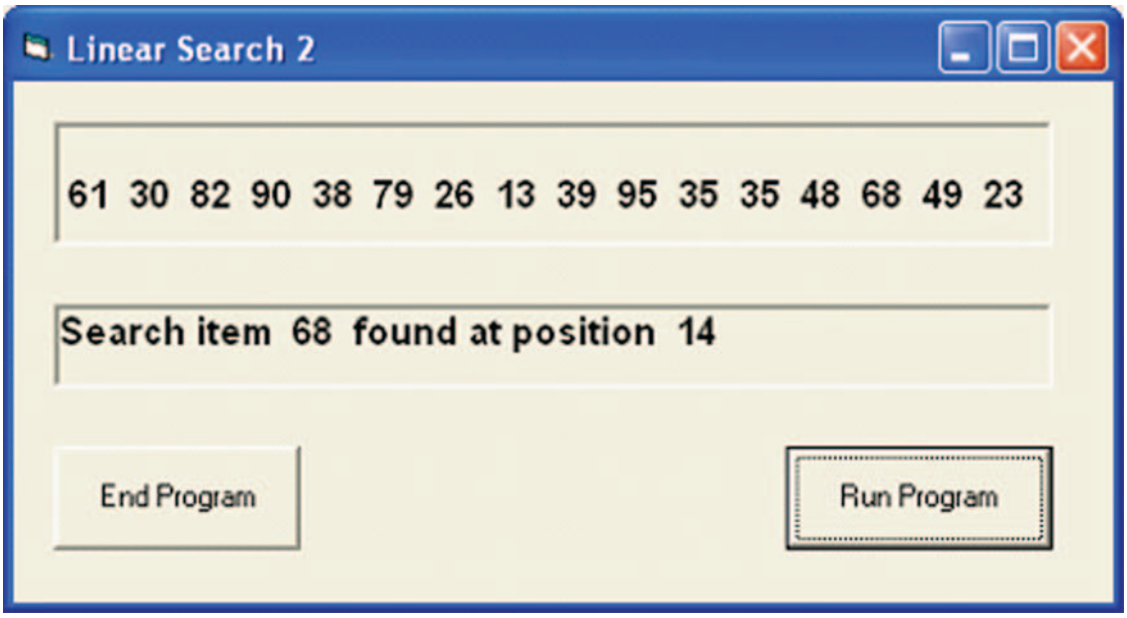


Figure 8.5:

The linear search is not the most efficient search strategy since each array element has to be compared with search key until a match is found. For example, for an array holding 1000 items, on average 500 comparisons will be made before the key is found.

If the data items are sorted into order, however, then the search time can be improved using a more complex algorithm called binary search. This is covered at Advanced Higher level.

Despite the large number of comparisons required to find an entry it is probably the fastest technique for small arrays. It is also the most simplistic in terms of coding. It may also be the only method to search larger, unordered tables of data.

Coding linear search

1. Code the example above.
2. Test that it works for a comprehensive range of data.
3. Store it in a module library for later use.

30 min

Searching a list of names

Adapt the previous program so that it that will search a list of names for a given name. You should consider the following tests where the name:

- does not appear in the list
- appears at the start of the list;
- appears at the end of the list;
- appears within the list.

8.5.3 Counting Occurrences

Programs often have to count occurrences. Examples include counting the number of:

- students who achieved particular marks in an examination
- rainfall measurements greater than a particular level
- words equal to a given search value in a text file.

The basic mechanism is simple:

1. a counter is established
2. a list is searched for the occurrence of a search value
3. every time the search value occurs, the counter is incremented

There is an interactivity online which you should attempt now.

You should notice that counting occurrences examines the entire list and so the algorithm is a variation of the linear search algorithm described above. The general counting occurrences algorithm is:

Counting occurrences - general algorithm

```
1.  counter = 0
2.  prompt the user for the search key
3.  set pointer to start of list
4.  do
5.  compare search key to list(position)
6.  if equal then
7.  increment count
8.  end if
9.  move to next position in the list
10. until end of list
11. report number of occurrences
```

Below is a Visual BASIC implementation.

```
Option Explicit
Dim SearchKey As Integer, Pointer As Integer, Fill As Integer
Dim Occurrence As Integer
Dim List(15) As Variant
Private Sub Command1_Click()

'Program Counting Occurrences
'29th February 2004
'Program by Fred Fink
```

```
'This program will perform a search on
'an array holding 16 random integers and
'output occurrences of an input value

Setup                                                    'Call procedures
Populate_List List()
Enter_search_item SearchKey
Search_for_item Pointer, SearchKey, Occurrence, List()

End Sub


Private Sub Setup()                                  'Initialise variables
  Randomize                                          'and clear output boxes
  PicList.Cls
  PicResult.Cls
  PicList.Print
  Pointer = 0
  Occurrence = 0
End Sub

Private Sub Populate_List(ByRef List())              'Fill array with
  Dim Fill As Integer                                '16 random numbers
  For Fill = 0 To 15
    List(Fill) = Int(49 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub

Private Sub Enter_search_item(ByRef SearchKey) 'User input search item
  SearchKey = InputBox("Input search item to count")
End Sub

Private Sub Search_for_item(ByVal Pointer, ByVal SearchKey,
ByVal Occurrence, ByRef List())
  Do
     If List(Pointer) = SearchKey Then
     Occurrence = Occurrence + 1
     End If
     Pointer = Pointer + 1
  Loop Until (Pointer > 15)
  PicResult.Print "Search item "; SearchKey; " found "; Occurrence;
"times in the list."
End Sub

Private Sub Command2_Click()
  End
End Sub
```

This file (Occurrences.txt), can be downloaded from the course web site.

In this case the array is filled with values from 1 to 49 in order to increase the chances of multiple occurrences.

The output of this can be seen in Figure 8.6

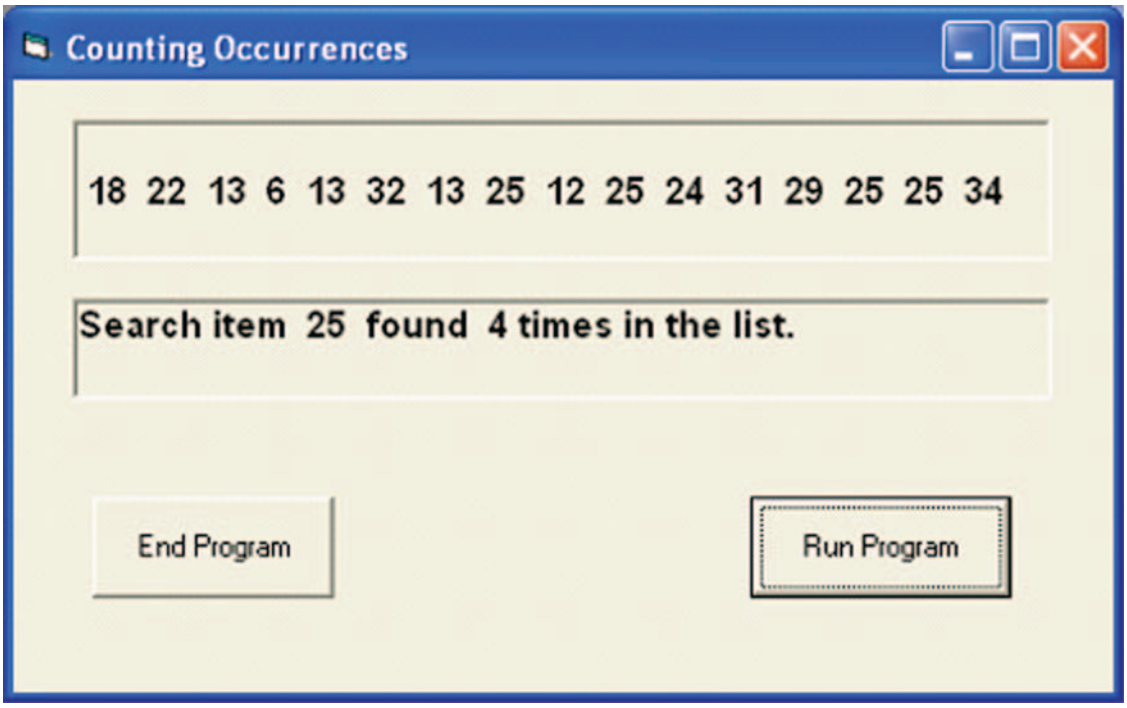


Figure 8.6:

Counting names in a list


45 min

Write a program that will count the number of times a search **name** appears in a list of names. Use the algorithm you have seen in the counting occurrences animation to help you. Remember to construct appropriate test cases in advance of coding. You should consider tests which check for:

- boundary conditions e.g. occurrence of the search value at the beginning or end of the list;
- no occurrences of the search value within the list;
- a single occurrence of the search value;
- multiple occurrences of the search value.

8.5.4 Finding Maximum

Computers are often used to find maximum and minimum values in a list. For example, a spreadsheet containing running times for videos might make use of a maximum algorithm to identify the video with the longest running time, or a minimum algorithm to identify the shortest running time. A database containing personal details of club membership might make use of maximum and minimum algorithms to identify the oldest or youngest member. You can think of a few more for yourself. Clearly these algorithms

are extremely useful and very widely used.

To find a maximum, we set up a variable which will hold the value of the largest item that has been found **so far**, usually the first data element. If an element in the array exceeds this working maximum, update the working maximum that value.

Such algorithms sometimes have to return the index number of the largest or smallest element, and sometimes the actual maximum or minimum value. The algorithms to return the maximum and minimum values are shown below:

Finding the maximum

```
1. set pointer to zero
2. set maximum to first item in list
3. Do
4. increment pointer
5. compare maximum to item at current position
6. if item > maximum then
7. set maximum to item
8. Loop until end of list
9. report maximum value
```

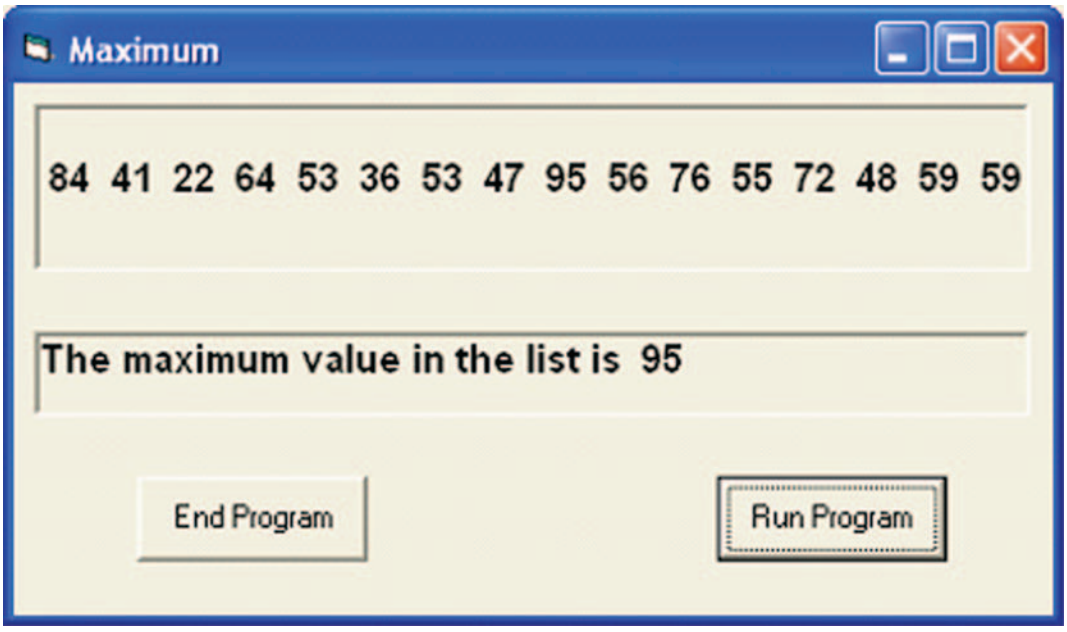
Online there is an interactivity you should attempt now.

The full Visual BASIC program for maximum is shown below:

```
Option Explicit
Dim SearchKey As Integer, Pointer As Integer, Fill As Integer
Dim Maximum As Integer
Dim List(15) As Variant
Private Sub Command1_Click()

'Program Maximum
'29th February 2004
'Program by Fred Fink

'This program will perform a search on
'an array holding 16 random integers and
'output the maximum value

Setup                                              'Call procedures
Populate_List List()
Search_for_max Pointer, Maximum, List()

End Sub


Private Sub Setup()                              'Initialise variables
  Randomize                                          'and clear output boxes
  PicList.Cls
```

```
  PicResult.Cls
  PicList.Print
  Pointer = 0
  Maximum = 0
End Sub

Private Sub Populate_List(ByRef List())        'Fill array with
  Dim Fill As Integer                                    '16 random numbers
  For Fill = 0 To 15
    List(Fill) = Int(100 * Rnd) + 1
    PicList.Print List(Fill);
  Next Fill
End Sub


Private Sub Search_for_max(ByVal Pointer, ByVal Maximum, ByRef List())
  Maximum = List(Pointer)
  Do
     Pointer = Pointer + 1
     If Maximum < List(Pointer) Then          'find maximum
       Maximum = List(Pointer)
     End If
  Loop Until (Pointer > 14)
  PicResult.Print "The maximum value in the list is "; Maximum
End Sub

Private Sub Command2_Click()
  End
End Sub
```

This file (Maximum.txt), can be downloaded from the course web site.



8.5.5 Finding minimum

Finding the minimum is almost exactly the same as finding the maximum; only 1 line of code needs to be changed in the algorithm (although for readability, you will want to change the variable name maximum to minimum).

Program to Find Minimum

1. Rewrite the algorithm form the previous page for Finding a Minimum.
2. Adapt the coding to match.
3. Test your new program.
4. Store it for future use.

8.6 Further activities on Standard Algorithms

Finding the maximum and minimum value in a list of integers

Write a program that will find the maximum and minimum values in a list of integers. Remember to construct appropriate test cases in advance of coding. You should consider the following tests:

- a normal test where maximum and minimum are as expected;
- a test where the maximum and minimum values are the same.

Sentence completion - algorithms

On the Web is a sentence completion task on algorithms. You should now complete this task.

8.7 Summary

The following summary points are related to the learning objectives in the topic introduction:

- modular program is very important in developing maintainable code;
- modular program involves - using procedures and functions to break up large blocks of code;
- avoiding the use of global variables;
- using parameter passing to pass data items between sub-programs;
- value parameters can only be passed IN to a procedure, and the main program variable cannot be changed;

- reference parameters are used where data must be passed OUT of a procedure; the main program variable can be changed;
- user-defined functions are an efficient and readable way of coding sub-programs which return a single value;
- you should be able to write pseudocode and high level language code for the following standard algorithms:
    1. linear search;
    2. count occurrences;
    3. find maximum;
    4. find minimum.

8.8 End of topic test

An online assessment is provided to help you review this topic.

Topic 9

End of Unit Test

Contents

An online assessment is provided to help you review this topic.

Glossary

Acceptance testing

Testing of software outside the development organisation and usually at the client site.

Adaptive maintenance

Takes place when a program's environment changes, for example a different operating system.

Algorithm

A detailed sequence of steps which, when followed, will accomplish a task.

Alpha testing

Testing of software within the development organisation.

Beta testing

Testing of software outside the development organisation using clients or selected members of the public.

Bottom-up design

A method of program refinement that starts with individual modules and builds them up into a complete program.

Boundary testing

Running a program with test data that represents the extreme upper and lower values. Within this range the program should operate normally.

Bugs

A bug is a program error.

Bytecode

This is produced by JavaScript and is a form of machine code that runs under the Java virtual environment. The latter is freeware and enables any computer to run Java programs

Client

The person or group that initiates the development process by specifying a problem .

Compiler

A program that translates a complete high level language program into an independent machine code program.

Concatenation

Joining of Visual BASIC string variables to make longer strings using the '&' operator.

Conditional loop

A control construct which allows a block of code to be repeated; the number of repeats is not known in advance by the programmer, but depends on a condition which may be true or false, often depending on user input.

Conditional loops

These loops test for a condition after each iteration, before performing a further loop.

Corrective maintenance

Correction of previously undetected errors during development that is now apparent after installation of the software on the client site.

COTS

Commercial Off The Shelf software. An alternative software development system that allows programmers to purchase ready-made software. Can be an expensive option.

Data

Unstructured information. A collection of numeric or alphanumeric characters which can be processed by a computer. Raw data is meaningless to people.

Database

An organised and structured collection of related data.

Data modelling

A process used in object oriented languages that identifies objects, how they relate to one another and their manipulation.

Debugging

The detection, location and removal of errors in a program.

Declarative language

Programmers use this type of language to specify what the problem is rather than how to solve it by writing code. The language uses facts and rules to express relationships.

Desk checking

Akin to a dry run where the running of a program is checked without a computer.

Development team

Generic description of the personnel involved in developing the software solution.

Dry run

A pen and paper exercise to debug a program.

efficient

an efficient program is one which does not make unnecessary demands on processor time, memory or other system resourses.

Event driven

A system that responds to an external event such as mouse click or a key press.

Event driven language

An event driven language that is designed to handle external events like interrupts, mouse clicks etc

Exceptions testing

Testing the robustness of a program by entering silly data - character data instead of numeric data, excessive values etc.

Executable code

Independent machine code that can be run without translation.

Exhaustive testing

Complete testing of a program under every conceivable condition. An expensive method time-wise.

Explicit declaration

Each variable, for example is declared unambiguously by the user so there is much less room for error in running programs Visual BASIC.

Feedback

A looping system where information is fed back in to a computer system. Previous output becomes new input.

Fit for purpose

The finished program runs to specification and is robust and reliable.

Fixed loop

A control construct which allows a block of code to be repeated a number of times set by the programmer in advance.

Fixed loops

These loops iterate a fixed number of times.

Function

A block of code like a procedure but a value is returned when the function is used.

Functional language

A language that utilises the evaluation of expressions rather than the execution of commands. It is based on the use of functions from which new functions can be created.

Functional specification

This will detail how the developed program will behave under specified conditions.

General purpose language

The language can be used to program solutions covering a broad range of situations.

High-level language

A language designed to be easily understood by programmers. They use commands and instructions based on English words or phrases.

Human computer interface

Allows the program to interact with the outside world. The interface is the only part of the program that users see.

Implicit declaration

If a variable, for example is not fully declared by the user then it is given default attributes by the Visual BASIC language.

Independent test group

Testing of software by a group out with the development team.

Inheritance

The sharing of characteristics between a class of object and a newly created sub class. This allows code re-use by extending an existing class.

Intermediate code

A form of compiled code that is specifically produced for a target computer.

Internal commentary

The use of comments within source code to describe what it does.

Internal documentation

The use of comments within source code to describe what it does.

Interpreter

A program that translates a high level program line by line, which it then tries to execute. No independent object code is produced.

Iterative

An iterative process is one that incorporates feedback and involves an element of repetition.

Jackson Structured Programming

A diagrammatic design method for small programs that focuses on sequence, selection and iteration.

Java

A language designed by Sun Microsystems. The language is portable because Java interpreters are available for a wide range of platforms.

Keyword

A reserved word with a special meaning in a computer language. For example for, if, dim in Visual BASIC.

Legal contract

A contract set up between client and development team, the details of which are set out in the requirements specification which becomes legally binding should anything go wrong.

Lexical analysis

Part of the compilation process where the source code is tokenised into symbols and stored in the symbol table.

Linear search

A standard algorithm that perform a sequential search on a list of data items.

Machine code

Native computer code that can be understood without translation.

Macro

A block of code that automates a repetitive task. Rather like a batch file they are normally created within an application then run by activating a key press combination or clicking on an icon.

maintainable

software which is written and documented in a way which makes it easy for programmers to correct errors, add new features or adapt the software.

Maintenance

The upkeep of a program by repair and modification.

Methodology

A technique involving various notations that enables the design of software to be implemented.

Module library

A module library includes code for standard algorithms that can be re-used by programmers.

Normal operation

Running of a program under expected normal conditions.

Object

A data item that can be manipulated by a computer system, for example a database record or a file.

Object code

The machine code produced by a compiler, ready for execution by a processor.

Object oriented design

A method that centres on objects and the operations that can be performed on them.

Object-oriented language

An object-oriented computer language like Java that uses objects rather than actions and data rather than logic. An object is represented by a class that can be extended to involve inheritance.

Optimised

Refinement of code to make it more efficient.

Parameter

An argument of a procedure or function that represents a local variable.

Parameter passing

The mechanism by which data is passed to and from procedures and the main program.

Perfective maintenance

Takes place when a system has to be enhanced in some way e.g. program run faster.

Portable

The ability of a program to run on different machine architectures with different operating systems.

Problem oriented

The focus is on the problem and how it is to be solved rather than on the hardware on which the program will run.

Problem specification

A document outline of what is to be solved in terms of programming a solution to a given problem.

Procedural language

Also known as imperative languages because the programs follow a sequence of steps until they terminate. The code is made up of procedures and functions.

Procedure

A block of code that, when called from within a program will perform a specific action.

Process

An activity that is performed by a piece of software,

Programming team

A section of the development team responsible for the coding, testing, implementation and maintenance of the software.

Project manager

A member of the development team who is responsible for the supervision of the project. The main tasks are to keep the project on schedule and within budget.

Pseudocode

A notation combining natural language and code used to represent the detailed logic of a program i.e. algorithmic notation.

RAD

Rapid Application Development. An alternative software development model that uses event driven languages for its implementation.

Recursion

A programming technique that is iterative in that a procedure or function can call itself. It is very demanding of computer memory.

Reference parameter

Here the address of the actual parameter is accessed by the formal parameter. Information is passed OUT from the procedure to the main program.

Reliable

A program is reliable if it runs well and is never brought to a halt by a design flaw.

Repetition

A process that repeats itself a finite number of times or until a certain condition is met.

Requirements specification

A document describing what the system must be able to do in order to meet user requirements.

Robust

A program is robust if it can cope with problems that come from outside and are not of its own making.

Scripting language

Used for writing small programs or scripts that enhances existing software. The best example is JavaScript which is used to enhance web pages.

Search key

The data item being searched for by a search algorithm.

Semantics

Semantics is the meaning of a statement in a given language.

Simulation

Replication of a process by computer that would not be possible to do manually. For example studying the projected traffic analysis of an airport or throwing a die many hundreds of times.

Software development environment

The high level language programming environment that offers tools and techniques to design and implement a software solution.

Software development process

A series of stages involving defined methods to produce a software project according to an initial specification.

Software engineering

A sphere of computing where the emphasis is on the development of high quality, cost effective software produced on schedule and within agreed costs.

Source code

The code for a program written in a high level language. This code is then translated into machine code.

Special purpose language

Languages designed for specific tasks such as prolog for artificial intelligence or C for writing operating systems.

Specification

A document outlining the program requirements set by the client.

SSADM

Structured Analysis and Design Model. An alternative to the waterfall model that deals only with the analysis and design phases of software development.

Standard algorithm

An algorithm that appears over and over again in many programs. Also called common algorithms.

Stepwise refinement

Similar to top-down design of sectioning a large and complex system into smaller and more easily manageable components.

Structure charts

A diagrammatic method of designing a solution to solve a software problem.

Structured data

Data that is organised in some way, for example an array or database.

Structured listing

Program listing clearly showing the modules involved complete with commentary and meaningful variable and procedure names.

Stub

A temporary addition to a program used to assist with the testing process.

Symbol table

Part of the compilation process where the tokens created by the lexical analysis phase are stored.

Syntax

Syntax means structure or grammar of a statement in a given language

Systems analyst

The person responsible for analysing and determining whether a task is suitable for pursuit using a computer. They are also responsible for the design of the computer systems.

Systems developer

Another name for a systems analyst.

Systems specification

An indication of the hardware and software required to run the developed program effectively. It will be the basis of subsequent stages which prepare a working program.

Technical guide

Documentation intended for people using a system containing information on how to install software and details system requirements such as processor, memory and backing storage.

Test data

Data that is used to test whether software works properly and that it is reliable and robust.

Testing

Running a program with test data to ensure a program is reliable and robust.

Test log

A record of how a program responds to various inputs.

Test plan

A strategy that involves testing software under verifying conditions and inputs.

Top-down design

A design approach of sectioning a large and complex system into smaller and more easily manageable components.

Trace facility

A method used to debug a program by tracing the change in values of the variables as the program is run.

Traditional model

An alternative name for the waterfall model that details the seven stages of program development.

Unusual user activity

Running a program with exceptional data.

User guide

A document intended for people using a system containing information on how to use the software.

Value parameter

Here a copy of the actual parameter is passed in to the formal parameter. Information is passed IN to the procedure form the main program.

Waterfall model

One of the earliest models for software development that incorporates 7 stages from analysis to implementation and maintenance.

Hints for activities

Topic 6: High Level Language Constructs 1

Calculating minutes

Hint 1: This problem naturally breaks up into four sections:

1. declare the variables;
2. get input;
3. calculate the minutes;
4. output results.

The input must consist of three numbers, the days, the hours and the minutes. So you need three variables to hold these figures. What will you name them? What type will they be (integer, real...)?

How do you work out the total number of minutes from these figures? You have to design a section of code to do this

Hint 2: There are 60 minutes in one hour and 24 hours in a day, hence 720 (24*60=1440) minutes in a day.

Hint 3: **For example:** days = 1, hours = 12, minutes = 15,
Total number of minutes = 2175.
days = 3, hours = 4, mins = 5
total minutes = 4565
days = 17, hours = 10, mins = 0
total mins = 25080

Calculating the number of digits in a number

Hint 1: It helps the user to know what type of number to type in, and what range of numbers is allowable *before* they type the wrong input! This should be in addition to any checks you do on the length of the number.

Calculating Leap Years

Hint 1: shows several years and indicates whether they are leap years or not. This data could be used to test your program. Table 9.1

Table 9.1: Calculation of leap years

Year	Is a Leap Year?
1900	No
1996	Yes
2000	Yes
2001	No

Answers to questions and activities

2 Features of Software Development Process

Revision (page 11)

Q1: c) Design, implementation, documentation, evaluation

Q2: d) User and Technical Guides

Q3: a) Program listing

Answers from page 13.

Q4: d) Analysis, design, testing, evaluation

Q5: a) The team may go back to an earlier stage to deal with a problem.

Q6: d) All of the above

Q7: b) Enable personnel to discuss progress

Answers from page 17.

Q8: a) top-down design

Q9: d) software specification

Q10: d) All of the above

Q11: b) Robust

Q12: a) a description of what the software must do

Answers from page 23.

Q13: c) It can make running a program a less irritable experience

Q14: c) Assembler

Q15: d) Source code

Answers from page 26.

Q16: b) Testing is done within the organisation

Q17: a) The program is tested by the clients

Q18: a) The input of unexpected data

Answers from page 30.

Q19: c) The program can cope with mistakes that the user might make

Q20: d) The program runs to specification

Q21: d) All of the above

3 Tools and techniques

Revision (page 37)

Q1: c) It is very useful in complex program designs

Q2: d) They represent the design in a visual way

Q3: b) English and high level language code

Answers from page 43.

Q4: a) Structure charts

Q5: b) It is breaking complex problems down into smaller units

Q6: c) Stepwise refinement

Q7: a) Think more about the solution to the problem

Q8: d) The designer can concentrate on a small part of the problem at a time

Preparing test data (page 44)

Q9: Sample solution:

1. 0 to 100
2. Loop
   Enter user input
   If user input invalid, then report error message
   Until input is valid
3. (in first 2 columns)
   5 valid data
   0 boundary / extreme data
   95 valid data
   100 boundary / extreme data
   -10 exceptional - should be rejected
   110 exceptional - should be rejected
   A exceptional - should be rejected
   10.5 may be valid or exceptional - specification does not make clear whether or not real numbers should be accepted.

Answers from page 44.

Q10: a) To determine that the system meets the specification

Q11: b) Output statements at key points in the code

Q12: c) Design input routines that will not crash when presented with unexpected data

Q13: a) They could remain hidden until the program is run under all conditions

4 Personnel

Answers from page 55.

Q1: c) The group who will purchase the software

Q2: d) Benefit the organisation in some way

Q3: d) The systems analyst is responsible for the entire project

Q4: a) Allow the systems analyst to produce a clear specification of the problem

Q5: b) The project manager

Answers from page 58.

Q6: d) All of the above

Q7: a) They report directly to the project manager at all stages of programming

Q8: b) Programmers will tend to test only within the functionality of their own code

Q9: d) The project manager

Q10: c) Testing is done by external groups on a variety of computer platforms

5 Languages and Environments

Revision (page 63)

Q1: d) all of the above

Q2: b) A compiler produces object code for a whole program in one operation

Q3: d) Syntax error

Answers from page 68.

Q4: b) languages have to be adapted so new versions are released

Q5: c) BASIC, Algol, Pascal, Comal

Q6: d) Moderation

Q7: c) Visual BASIC

Q8: d) They are low level languages

Answers from page 77.

Q9: a) Programs have no pre-defined pathway

Q10: d) They describe a problem rather than how to solve it

Q11: c) Macros automate repetitive tasks

Q12: d) Changing audio CDs

Answers from page 80.

Q13: c) A compiler creates an independent machine code program

Q14: a) Looping structures have to be interpreted each time they are entered

Q15: a) Computers can only understand machine code

6 High Level Language Constructs 1

Revision (page 83)

Q1: d) Answer = 5 + 8 * (3 - 2)

Q2: c) 6

Q3: d) All of the above

Answers from page 99.

Q4: c) 17.5%Vat

Q5: a) It makes it easier for programmers to locate and fix errors

Q6: d) Integer

Q7: c) Integer (long)

Answers from page 103.

Q8: The variable `number2` has not been declared. There will be no output since the variable sum has no value and has not been declared. Also the `Print` statement is wrong.

Answers from page 111.

Q9: c) Global

Q10: d) All three statements above

Q11: b) They are hidden from other procedures and functions

Q12: c) The extent to which the variable can be 'seen' by the rest of the program

Q13: d) It can have only the values true or false

Answers from page 113.

Q14: c) 1

Q15: a) 3.5

Q16: b) 3

Using the logical AND operator in an if statement (page 126)

If(number1>10) AND (number2>10) Then Print . . .

Answers from page 129.

Q17: d) NOT

Q18: a) 11

Q19: c) The execution of program statements in order, from beginning to end

Q20: b) The expression is TRUE when both conditions being tested are TRUE

Q21: c) Assignment

Outwith range (page 129)

Answer to come

7 High Level Language Constructs 2

Revision (page 147)

Q1: c) Data items of the same type are grouped together

Q2: d) Days(3) = "Wednesday"

Q3: c) 1, 8, 3, 9, 5, 6

Answers from page 157.

Q4: d) Control loop variable is increasing in value by a constant amount determined by the programmer

Q5: d) For loopCounter = (3*4.16) to (5*5.6) step 1

Q6: a) Calculating the total number of marks entered at a keyboard

Q7: a) Triangle

Q8: c) 18

Answers from page 165.

Q9: The value of `n` within the loop is never incremented. It will therefore always have a value of 0, which is less than 100, and hence this will produce an infinite loop which will never terminate. The program will therefore never end.

The code could be altered to include an increment, i.e.

```
n = 0
do while n < 100
    value = n*n
    n = n + 1
loop
```

Q10: There is no `loop` to end the `do..while`

Add loop at the end of the code:

```
i = 1
do while i <= 10
    print(i)
    i = i + 1
loop
```

Q11: A typical solution to the problem is shown here.

```
dim i as integer, sum as integer
sum = 0
i = 0
do while i <= 20
    sum = sum + i
    i = i + 1
loop
```

You should obtain a value of 210 for the sum of all numbers between 0 and 20 inclusive.

Answers from page 171.

Q12: c) The loop need not be entered if the condition fails at the start

Q13: a) Selection

Q14: b) 1 2 3 4 5

Q15: b) 1

Q16: d) Any one of the above

Indexing arrays (page 178)

Q17: c) array = [0, 3, 0, 0]

Q18: d) array = [7, 3, 0, 0]

Q19: c) 2

Q20: a) 3

Q21: c) array = [8, 12, 5, 19, 3, 0, 7, 52]

Q22: c) array = [8, 23, 5, 9, 3, 4, 7, 52]

Q23: array = [0, 9, 3, 0]

Q24: array = [0, 7, 0, 0, 6, 0, 0, 3, 2, 0]

Q25: The program fragment which will help you to do this is:

```
for i=0 to 9
  myarray(i) = i
```

Answers from page 180.

Q26: d) All of the above options

Q27: b) A for loop

Q28: d) 0 4 6 2 5 7

Q29: a) value_1(0) = "Hello"

Q30: b) 10

Heads and Tails Program (page 185)

The full Visual BASIC program is shown in Code 38.

```
Option Explicit
```

```
Dim HeadsTails(1 To 1000) As Integer
Private Sub Command1_Click()
  '20th February
  'Program by Fred Fink

  'This program simulates the tossing of a coin


  Dim toss As Integer, heads As Integer, tails As Integer
  Dim count As Integer

  Randomize
  PicHeads.Cls
  PicTails.Cls
  PicTosses.Cls
  count = InputBox("Enter number of tosses")

  For toss = 0 To count-1
    HeadsTails(toss) = Int((2 * Rnd) + 1) 'Generate numbers
  Next toss

  For toss = 0 To count-1                  'Count 1's and 2's
     If HeadsTails(toss) = 1 Then
     heads = heads + 1
     Else
     tails = tails + 1
     End If
  Next toss

  PicHeads.Print heads                    'Display results
  PicTails.Print tails
  PicTosses.Print count
End Sub
```

This file (HeadsTails.txt), can be downloaded from the course web site.

Code 38

Note that to make the program easier to follow two `for..next` loops are used but they could be combined into a single loop.

Coding the Palindrome Checker (page 189)

When you do code it successfully, you will have results like those shown in Figure 9.1

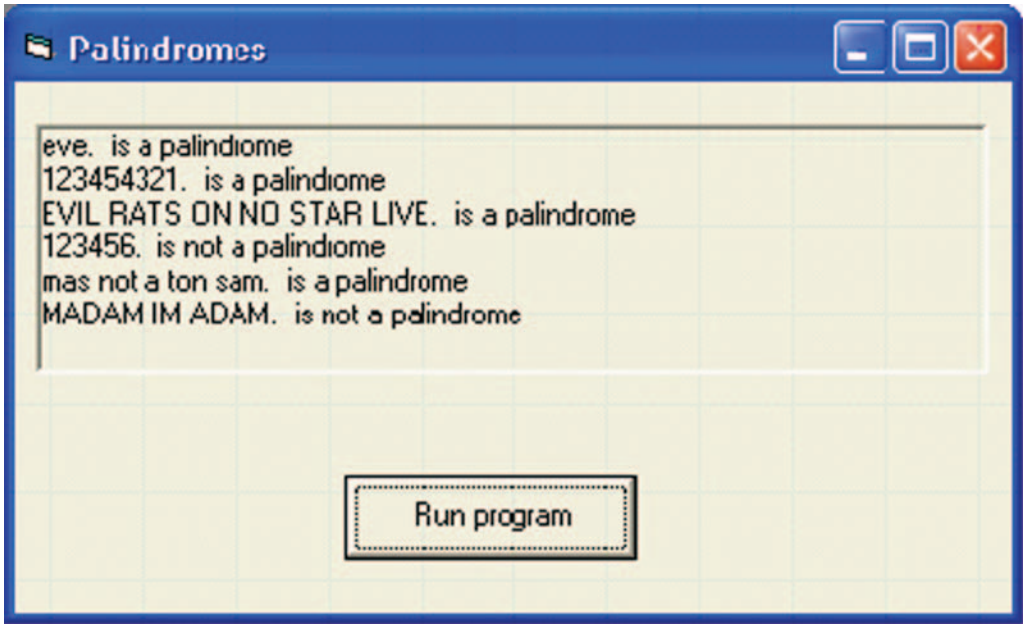


Figure 9.1:

There are other methods, probably much simpler, of writing programs to test for palindromes. However it is essential that you understand the use of arrays since they are important data structures in all fields of computing. You will see more in the use of arrays in the final topic.

8 Procedures, Functions and Standard Algorithms

Revision (page 193)

Q1: b) That the input data is within specified limits

Q2: d) All of the above

Q3: b) A sequence of instructions that can be used to solve a common problem

Program with parameters (page 197)

```
Private Sub Get_valid_number
 ' input validation
 Dim Number As Integer
 Dim Inrange As Boolean
Do
 Number =  InputBox("Enter a number in the range 1 to 30")
 Inrange = (Number>=1) AND (Number<=30)
 If not inrange Then MsgBox("Out of range.  Try again.")
Loop Until Inrange
End Sub
```

This file (parameters.txt), can be downloaded from the course web site.

byVal or byRef (page 203)

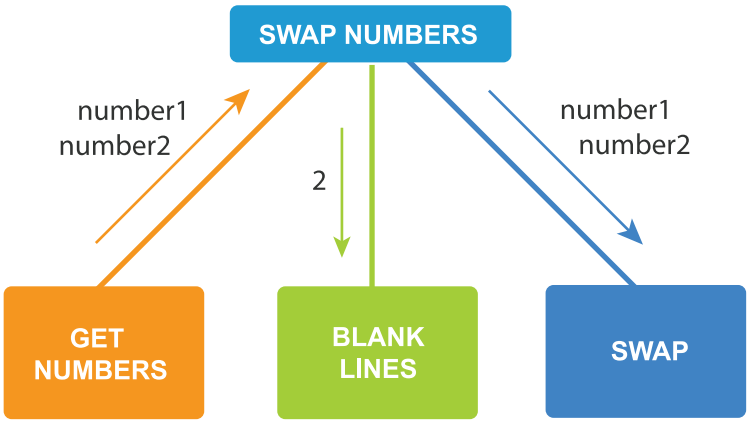
Q4: b) byVal

Q5: a) byRef

Q6: a) byRef

Q7: a) byRef

Designing the solution (page 210)




```
1 getNumbers OUT Number 1, Number 2
2  Blanklines IN 3
3 Swap IN Number1, Number 2
```

Answers from page 211.

Q8: b) Repetition of lines of code are avoided

Q9: d) A footer

Q10: b) byRef is used for IN-OUT parameters

Q11: b) Anything that happens to the formal parameter happens to the actual parameter.

Q12: c) 5 6 15 6

Turning a procedure into a function (page 215)

```
function getValidNumber as integer
Dim numberOk as boolean
Dim number as integer
  do
    number = Inputbox("Input a number")
    numberOK = number > 0
    if not numberOK then
      Print("The number must be greater than 0")
    end if
    loop until numberOK
    getValidNumber = number
end function
```

Answers from page 215.

Q13: c) A function returns a single value.

Q14: c) Square

Q15: b) y = ABS(-5)

Program to Find Minimum (page 226)

Expected answer (pseudocode)

1. set pointer to zero
2. set minimum to first item in list
3. Do
4. increment pointer
5. compare minimum to item at current position
6. if item < minimum then
7. set minimum to item
8. Loop Until end of list
9. report minimum value

Your result should resemble the images below.